robotron

ANWENDERDOKUMENTATION

BASIC-Interpreter

für SIOS 1526

Systemunterlagen MOS
Dokumentation Anwenderdokumentation
Stand: September 1984

A N W E N D E R D O K U M E N T A T I O N

BASIC - Interpreter
für SIOS 1526

3. Auflage

VEB Robotron - Buchungsmaschinenwerk
Karl-Marx-Stadt

Die vorliegende Anwenderdokumentation entspricht dem Stand vom September 1984

Nachdruck, jegliche Vervielfältigung dieser Unterlage oder Auszüge daraus sind unzulässig.

Die Ausarbeitung dieser Dokumentation erfolgte durch ein Kollektiv der Abteilung Software- Entwicklung des VEB Robotron- Buchungsmaschinenwerk Karl-Marx-Stadt.

Herausgeber:

VEB Robotron- Buchungsmaschinenwerk
Karl-Marx-Stadt

9010 Karl-Marx-Stadt
PSF 129

Vorwort

Die vorliegende Anwenderdokumentation umfaßt die:

- Anleitung zur Bedienung des BASIC-Interpreters sowie
- BASIC- Sprachbeschreibung.

Das BASIC- System steht für die Bürocomputer A 5120/ A 5130 zur Verfügung. Es arbeitet unter Regie des Betriebssystems SIOS 1526.
In einem ersten Teil dieser Schrift ist die Arbeit mit dem BASIC- System dargestellt, insbesondere werden die Kommandos zur Bedienung beschrieben.
Im zweiten Teil ist die problemorientierte Sprache BASIC erläutert. Die Syntax wird in einer übersichtlichen Form angegeben. Die Semantik wird verbal beschrieben. Beispiele in Form von Anweisungsfolgen demonstrieren die jeweiligen Sprachelemente.

I N H A L T S V E R Z E I C H N I S

Teil I BASIC - Kommunikationssystem

1 . Einführung
2 . Allgemeines zur Arbeit mit dem BASIC- System
2 .1 . Sondertasten
2 .2 . Bereitschaftszeichen
2 .3 . Neustart, Wiederstart und Beendigung einer BASIC- Sitzung
2 .3 .1 . Neustart des BASIC- Systems
2 .3 .2 . Beendigung einer BASIC- Sitzung

2 .4 . Verbesserung von Tippfehlern
2 .5 . BASIC- Kommandos und BASIC- Anweisungen
2 .5 .1 . Kommandos
2 .5 .2 . Anweisungen
2 .5 .3 . Fehlermeldungen
2 .5 .4 . Änderungen oder Löschen einer Anweisung
2 .6 . BASIC- Programme
2 .7 . Benutzerarbeitsbereich
2 .8 . Auflisten eines Programms
2 .9 . Starten eines Programms
2 .10. Löschen eines Programms
2 .11. Dokumentation eines Programms
3 . BASIC- Kommandos
3 .1 . Programmausführungskommandos
3 .1 .1 . RUN
3 .1 .2 . XEQ
3 .1 .3 . Wiederaufnahme der Programmausführung durch CONTINUE, STEP, RUN
3 .1 .4 . QUIT
3 .2 . Editierkommandos
3 .2 .1 . LIST
3 .2 .2 . NEW
3 .2 .3 . DELETE

3 .2 .4 . RENUMBER
3 .2 .5 . SIZE
3 .2 .6 . CLEAR
3 .3 . Diskettenbezogene Kommandos
3 .3 .1 . SAVE
3 .3 .2 . ASAVE
3 .3 .3 . RSAVE
3 .3 .4 . GET
3 .3 .5 . XEQ
3 .3 .6 . APPEND
Teil II BASIC - Sprachbeschreibung

4 . Die Grundelemente der Sprache BASIC
4 .1 . Die Grundsymbole von SIOS- BASIC
4 .2 . Konstanten
4 .2 .1 . Zahlkonstanten
4 .2 .2 . Zeichenketten
4 .3 . Variablen
4 .4 . Funktionen
4 .5 . Operatoren
4 .6 . Ausdrücke und deren Auswertung
5 . Anweisungen
5 .1 . Zuweisungsanweisung
5 .2 . END/ STOP- Anweisung
5 .3 . Schleifenanweisung: FOR...NEXT
5 .4 . GOTO/ ON...GOTO- Anweisungen
5 .5 . GOSUB...RETURN- Anweisungen
5 .6 . Bedingte Anweisungen: IF...THEN
5 .7 . INPUT- Anweisung
5 .8 . PRINT- Anweisung und ":"
5 .9 . READ/DATA/RESTORE- Anweisungen
5 .10. Kommentare: REM- Anweisung und "\"
5 .11. RANDOMIZE- Anweisung
5 .12. SYSTEM- Anweisung
6 . Zahlenwerte und Wertarten
6 .1 . Typspezifikation

6 .2 . Form von Zahlenwertkonstanten
6 .2 .1 . Ganze Zahlen
6 .2 .2 . Gleitkommazahlen
6 .3 . Zahlenwert- Ausdrücke
6 .4 . Zahlenwert- Ausdrücke als Bedingungen
6 .5 . Zahlenwertzuweisung
6 .6 . Eingabe von Zahlenwerten
6 .7 . Ausgabe von Zahlenwerten
6 .8 . Zahlenwertfunktionen
6 .8 .1 . ABS- Funktion
6 .8 .2 . ATN- Funktion
6 .8 .3 . COS- Funktion
6 .8 .4 . EXP- Funktion
6 .8 .5 . INT- Funktion
6 .8 .6 . LOG- Funktion
6 .8 .7 . RND- Funktion
6 .8 .8 . SGN- Funktion
6 .8 .9 . SIN- Funktion
6 .8 .10. SQR- Funktion
6 .8 .11. TAN- Funktion
7 . Felder
7 .1 . DIM- Anweisung
7 .2 . Speicherung von Daten in Feldern
7 .3 . Ausgabe von Daten aus Feldern
8 . Zeichenketten
8 .1 . Zeichenketten- Konstanten
8 .2 . Zeichenketten- Variablen
8 .3 . DIM- Anweisung mit Zeichenketten
8 .4 . Zeichenketten- Ausdrücke
8 .5 . Zeichenkettenbezogene Funktionen
8 .5 .1 . CHR$- Funktion
8 .5 .2 . ASC- Funktion
8 .5 .3 . LEN- Funktion
8 .5 .4 . POS- Funktion
8 .5 .5 . VAL- Funktion
8 .5 .6 . STR$- Funktion

8 .5 .7 . LEFT$- Funktion
8 .5 .8 . RIGHT$- Funktion
8 .5 .9 . SEG$- Funktion
8 .6 . Vergleich von Zeichenketten
8 .7 . Zeichenkettenzuweisung
8 .8 . Zeichenketteneingabe- Zuweisung
8 .9 . Zeichenketteneingabe durch LINPUT- Anweisung
8 .10. Zeichenkettenausgabe durch PRINT- Anweisung
8 .11. Zeichenketten in READ/DATA/RESTORE- Anweisung
9 . Benutzerdefinierte Funktionen
9 .1 . Einzeilige Funktionen
9 .2 . Mehrzeilige Funktionen
9 .3 . Aufruf einer benutzerdefinierten Funktion
10. Dateien
10.1 . Öffnen von Dateien: FILE- Anweisung
10.2 . Schließen von Dateien: CLOSE- Anweisung
10.3 . Löschen von Dateien: ERASE- Anweisung
10.4 . TRUNCATE- Anweisung
10.5 . SPACE- Anweisung
10.6 . Datei- Zugriff
10.6 .1 . Sequentielles READ/INPUT/LINPUT auf Dateien
10.6 .2 . Sequentielles PRINT und WRITE
10.6 .3 . RESTORE- Anweisung für Dateien
10.6 .4 . READ/INPUT/LINPUT auf wahlfreie Dateien
10.6 .5 . PRINT und WRITE auf wahlfreie Dateien
10.7 . EOF- Funktion
11. Formatierte Ausgabe
11.1 . PRINT USING- Anweisung
11.2 . Format- Beschreibungen
12. Überlagerung von Programmen
12.1 . CHAIN- Anweisung
12.2 . COM- Anweisung
13. Im Tischrechner- Modus ausführbare Anweisungen
14. Aufruf von Assemblerunterprogrammen

Anhang A: KOI-7-BIT-CODE
Anhang B: SIOS- BASIC- Anweisungen (Zusammenfassung)
Anhang C: SIOS- BASIC- Kommandos (Zusammenfassung)
Anhang D: SIOS- BASIC- Funktionen (Zusammenfassung)
Anhang:E: Liste der Fehlernummern und Erläuterung

1 . Einführung

Eine der ersten und inzwischen am weitesten verbreiteten Dialogsprachen ist BASIC (Beginner's All purpose Symbolic Instruction Code). BASIC ist eine einfache Programmiersprache, die besonders für wissenschaftlich-technische Problemstellungen geeignet ist. Es lassen sich aber auch eine Reihe von kommerziellen Aufgabenstellungen lösen. Dialogsysteme besitzen neben der Problemsprache, in der die Aufgabe formuliert wird, eine Kommandosprache, die den Aufbau, die Testung, Änderung und Abarbeitung im Dialogbetrieb ermöglicht. Die vorliegende Schrift will, unterstützt von Beispielen, den Sprachumfang, der der Implementierung auf den Bürocomputern A 5120 bzw. A 5130 entspricht, vorstellen und die verwendete Kommandosprache erläutern.
Die Ausdrucksmittel der Sprache sind so angelegt, daß die Baugruppen Bildschirm, Folienspeicher, Drucker und Tastatur bedient werden. Die Geräte sollten mit 64 K-RAM-Speicher ausgestattet sein.
Minimal sind 54 K RAM erforderlich.

Die Dateien werden in einem Format gemäß der KROS Nr. 5108/01 bzw. 5110/01 aufgezeichnet. Damit sind sie kompatibel zu anderen Dateien, die durch SIOS verarbeitet werden (siehe Systemhandbuch, Teil III - 2.1.). BASIC beinhaltet eine BCD- Arithmetik mit einer Genauigkeit von 13 Dezimalstellen.

2 . Allgemeines zur Arbeit mit dem BASIC- System

Dieser Abschnitt beschreibt, wie man eine BASIC- Sitzung beginnt und beendet, wie man Kommandos und Anweisungen eingibt und Verbesserungen vornimmt. Einige einfache Beispiele dienen der Illustration. Weiterhin werden Sondertasten beschrieben, denen eine besondere Funktion in SIOS- BASIC zugeordnet ist. Außerdem wird einiges zum Starten, Auflisten, Löschen und zur Dokumentation

eines Programmes angeführt.

2 .1 . Sondertasten

ET1 — Muß nach jedem Kommando und jeder Anweisung gedrückt werden. Die Tastenbetätigung beendet die Zeile und der Cursor kehrt in die erste Druckposition zurück.

↵ — Veranlaßt die Einfügung eines Leerzeichens in den Text. Der Cursor kommt auf die erste Druckposition der nächsten Zeile.

← — Löscht das jeweils letzte eingegebene Zeichen. Der Cursor wird für jedes gelöschte Zeichen um eine Position zurückgesetzt. (Taste Rückschritt)

CE — Entwertet die ganze angefangene Zeile. (Taste Löschen Eingabe)

DEL — Entwertet die laufende Zeile, wenn sie während der Eingabe eines Befehls oder einer Anweisung gedrückt wird und beendet das augenblicklich laufende Programm, wenn sie im Rechen- Modus gedrückt wird.

? — Bewirkt die Unterbrechung bzw. Fortsetzung einer Ausgabe, wenn die Taste gedrückt wird.

ET2 — Tabulation des Cursors (8-er Schritte).

2 .2 . Bereitschaftszeichen

SIOS- BASIC verwendet zwei Bereitschaftszeichen. Damit wird der Anwender informiert, daß entweder bestimmte Eingaben erwartet werden oder daß bestimmte Aktionen beendet sind.

* — Das Systemeingabe- Bereitschaftszeichen des SIOS- BASIC- Interpreters. Es wird ein

BASIC- Kommando oder eine BASIC- Anweisung erwartet.

? Das Anwendereingabe- Bereitschaftszeichen des SIOS- BASIC- Interpreters. Es wird eine Anwendereingabe während der Ausführung einer INPUT- Anweisung erwartet.

2 .3 . Neustart, Wiederstart und Beendigung einer BASIC- Sitzung

2 .3 .1 .Neustart des BASIC- Systems

Die Aktivierung des Systems erfolgt dadurch, daß das Betriebssystem SIOS und der BASIC- Interpreter von der Systemdiskette geladen werden.
Folgende Arbeitsschritte sind zum Start des Systems nötig:

(1) Einlegen der Systemdiskette (mit Betriebssystem und BASIC- Interpreter) in ein beliebiges Laufwerk

(2) Einschalten des Gerätes durch Betätigung der Netztaste (durch Startprogramm wird Betriebssystem und BASIC- Interpreter automatisch geladen)

(3) Das Betriebssystem SIOS meldet sich in folgender Form und gibt damit seine Bereitschaft zur Entgegennahme eines Kommandos an:

```
MONITOR_=====_SIOS_1526/0.6_(M)
```

(4) Zum Aufruf von BASIC gibt man nach einem Neustart folgendes ein:

```
RUN (ET1)
```

(5) Das BASIC- System meldet sich mit folgender Ausschrift:

```
BASIC 1520(SIOS) V 0.3
==============================================
```

(6) Auf der nächsten Zeile folgt das Systemeingabe-Bereitschaftszeichen *

Nun können BASIC- Kommandos und BASIC- Anweisungen eingegeben werden. Jedes Kommando oder jeder Befehl wird durch das * - Zeichen am Anfang einer Zeile bestätigt.

2 .3 .2 . Beendigung einer BASIC- Sitzung

Wenn der Anwender fertig ist, gibt er die Kontrolle an SIOS mittels des QUIT- Kommandos zurück.
Um BASIC zu verlassen, gibt man folgendes ein:

```
*QUIT
```

SIOS signalisiert die Rücknahme der Kontrolle durch die Meldung mit der Systemzeile:

```
MONITOR␣=====␣SIOS␣1526/0.6␣(M)
```

2 .4 . Verbesserung von Tippfehlern

Verbesserungen sind während der Eingabe einer Zeile möglich, solange die ET1- Taste noch nicht gedrückt wurde. Die |← Funktionstaste wird benutzt, um wenige, gerade eingegebene Zeichen zu verbessern. Die CE Funktionstaste wird zur Entwertung der gesamten Zeile benutzt; es wird dann neu mit der Zeile begonnen.

Angenommen,
der Anwender verschreibt sich
beim Kommando RUN * RUM

die Funktionstastenbetätigung
|← löscht das letzte Zeichen * RU

Der Anwender gibt nun das richtige Zeichen ein und beendet die Zeile. Wenn ET1 gedrückt wird, ist die Zeile richtig eingegeben. * RUN

Falls mehrere Zeichen nach dem Fehler eingegeben wurden, muß die Funktionstaste |← je einmal für jedes zu löschende Zeichen gedrückt werden.

```
* 10  IXPUT
* 10  I          (Merke: |← Taste wurde 4x gedrückt)
* 10  INPUT
```

in diesem Fall wurden vier Zeichen gelöscht.
Eine andere Methode ist die Benutzung der CE Funktionstaste zur Löschung einer Zeile. Dabei ist diese Taste vor Betätigung der ET1- Taste zu drücken.
Zur Entwertung der Zeile wird die * 10 IXPUT
Taste CE betätigt. Der Anwender *
gibt die Zeile erneut ein. * 10 INPUT

2.5. BASIC- Kommandos und BASIC- Anweisungen

2.5.1. Kommandos

SIOS- BASIC- Kommandos veranlassen den SIOS- BASIC- Interpreter, bestimmte Kontroll- Funktionen durchzuführen. Kommandos unterscheiden sich grundsätzlich von Anweisungen, die benutzt werden, um ein Programm in der SIOS- BASIC- Sprache zu schreiben. Ein Kommando veranlaßt den Interpreter, eine Aktion sofort durchzuführen,

während eine Anweisung normalerweise nur während des Laufes eines Programms in eine Aktion umgesetzt wird. Ähnlich wie Kommandos können manche Anweisungen sofort ausgeführt werden.
Jedes SIOS- BASIC- Kommando kann im Anschluß an das BASIC- Bereitschaftszeichen "*" gegeben werden. Jedes Kommando ist ein einziges Wort. Wenn es falsch geschrieben wird, gibt der Bürocomputer eine Fehlermeldung. Einige Kommandos haben Parameter zur näheren Beschreibung der Kommando- Operation.
So ist zum Beispiel QUIT ein Kommando, daß den Abschluß einer BASIC- Sitzung und Rückkehr zum Betriebssystem signalisiert. Es hat keine Parameter. Ein anderes Kommando, LIST, druckt das gerade eingegebene Programm aus. Es kann Parameter haben, um anzugeben, daß nur ein bestimmter Teil des Programms gelistet werden soll.

/

2 .5 .2 . Anweisungen

Anweisungen werden benutzt, um ein SIOS- BASIC- Programm zu schreiben, das später ausgeführt wird. Jede Anweisung führt dann eine bestimmte Funktion aus. Jede Anweisung, die einmal eingegeben wurde, wird Bestandteil des augenblicklichen Programms und wird beibehalten, bis es explizit gelöscht wird oder der Benutzer mittels QUIT aussteigt.
Einer Anweisung in einem BASIC- Programm geht stets eine Anweisungsnummer voraus. Diese Nummer ist eine ganze Zahl zwischen 1 und 9999. Die Anweisungsnummern geben an, in welcher Reihenfolge die Anweisungen ausgeführt werden. Anweisungen werden vom BASIC- System der Größe der Anweisungsnummern nach geordnet, von der kleinsten bis zur größten Nummer. Weil diese Reihenfolge vom Interpreter eingehalten wird, ist es nicht nötig, daß der Benutzer die Anweisungen auch in der Reihenfolge ihrer

Ausführung eingibt, solange die Nummern sich in dieser Reihenfolge befinden.
Im Anschluß an die Eingabe jeder Anweisung muß ET1 gedrückt werden, um dem Interpreter mitzuteilen, daß die Anweisung vollständig ist. Der Interpreter erzeugt ein Zeilenendezeichen und gibt das Bestätigungs- Zeichen "*" in die nächste Zeile und signalisiert so die Annahme der Anweisung. Falls die eingegebene Anweisung fehlerhaft ist, bringt der Bürocomputer eine Fehlermeldung.
SICS- BASIC- Anweisungen haben ein freies Format. Das heißt, daß Leerzeichen ignoriert werden.
Zum Beispiel sind alle diese Anweisungen äquivalent:

```
*20 LET B7=25
*20LETB7=25
*20 L E T B 7 = 2 5
*20   LET  B7  =  25
```

2 .5 .3 . Fehlermeldungen

Wird in einer Zeile ein Fehler gemacht, und die Zeile mit ET1 abgeschlossen, erzeugt der Interpreter eine Meldung. Die Meldung besteht aus dem Wort ERR gefolgt von einer Nummer, die die Art des Fehlers anzeigt. Anhang E enthält eine Liste der Nummern der Fehler und Warnungen und der jeweiligen Bedeutung.

2 .5 .4 . Änderung oder Löschen einer Anweisung

Wenn ein Fehler bemerkt wird, bevor die ET1- Taste gedrückt wurde, kann der Fehler mittels ← Taste behoben oder die ganze Zeile durch ⊢ Taste entwertet werden (siehe Abschnitt 2.4.). Nach dem Abschluß der Zeile durch ET1 kann der Fehler durch Löschung oder Ändern der Anweisung verbessert werden.
Zur Änderung der Anweisung wird einfach die Anweisungsnummer erneut eingegeben, gefolgt von der verbesserten

Anweisung.

Um die folgende Anweisung zu ändern:
```
*20 LET B7=25
```
tippt man:
```
*20 LET B7=37
```

Eine solche Änderung kann zu jeder Zeit vor Ausführung des Programms gemacht werden.
Zur Löschung einer Anweisung wird die Anweisungsnummer, gefolgt von ET1 eingegeben.

Anweisung 20 wird gelöscht durch
```
*20
```

Das in Abschnitt 3.2.3. beschriebene DELETE- Kommando ist nützlich zur Löschung einer Gruppe von Anweisungen.

2 .6 . BASIC- Programme

Jede ausführbare Anweisung oder Gruppe von Anweisungen bilden ein Programm.
Ein Programm kann aus nur einer Anweisung bestehen.

Dies ist ein Beispiel für ein Programm mit nur einer Anweisung:
```
*100 PRINT "5 x 10 = ";5x10
```

100 ist die Anweisungsnummer. PRINT ist das Schlüsselwort, das dem Interpreter die Art der auszuführenden Aktion mitteilt. In diesem Fall wird die Zeichenkette "5 x 10 =" und der Wert des folgenden Ausdrucks ausgegeben. 5 x 10 ist ein arithmetischer Ausdruck, der durch den Interpreter ausgewertet wird. Normalerweise enthält ein Programm mehr als eine Anweisung.

Diese vier Anweisungen sind ein Programm:
```
*10 INPUT A,B
*20 LET C=A+B
*30 PRINT
*40 PRINT A;" +";B;" =";C
```

Dieses Programm, das die Summe zweier Zahlen berechnet,

ist in der Reihenfolge seiner Ausführung gezeigt. Es könnte in beliebiger Reihenfolge eingegeben werden, wenn die jeder Anweisung zugeordneten Anweisungsnummern nicht geändert werden.

Dieses Programm läuft genau wie das obige ab.

```
*20 LET C=A+B
*10 INPUT A,B
*30 PRINT
*40 PRINT A;" +";B;" =";C
```

Es ist generell sinnvoll, die Anweisungen mit der Schrittweite 10 durchzunumerieren. Dies erlaubt die spätere Einfügung von zusätzlichen Anweisungen bei Bedarf.

2.7. Benutzerarbeitsbereich

Wenn Anweisungen am Bürocomputer eingegeben werden, werden sie Bestandteil des Benutzer- Arbeits- Bereiches. Alle Anweisungen im Benutzer- Arbeits- Bereich zusammen bilden das aktuelle Programm.
Jede Anweisung im Benutzer- Arbeits- Bereich kann bearbeitet oder verändert werden; die neue Anweisung ersetzt dann die vorige Version im Benutzer- Arbeits- Bereich.
Wenn der Benutzer mittels des QUIT- Kommandos BASIC verläßt, wird der Arbeits- Bereich gelöscht.

2.8. Auflisten eines Programms

Zu jedem Zeitpunkt während der Eingabe eines Programms kann das LIST- Kommando benutzt werden, das eine Auflistung der vom Bürocomputer akzeptierten Anweisungen produziert. LIST veranlaßt die Ausgabe einer Auflistung des aktuellen Programms.
Nach Löschung oder Änderung einer Zeile kann LIST zur Kontrolle verwendet werden, ob die Löschung oder Änderung vollzogen wurde.

```
                        *10 INPUT A,B
                        *20 LET C=A+G
Während der Eingabe     *20 LET C=A+B
wurde eine Verbesse-    *30 PRINT
rung vorgenommen:       *40 PRINT A;" +";B;" =";C

                        *LIST
                          10 INPUT A,B
Zur Kontrolle, ob die     20 LET C=A+B
Verbesserung durchge-     30 PRINT
führt wurde, wird das     40 PRINT A;" +";B;" =";C
Programm gelistet:      *
```

Anmerkung:
Das Bereitschafts- Zeichen "*" wird in der Auflistung nicht ausgegeben. Aber es wird nach dem Ende der Auflistung generiert,und zwar als Hinweis, daß das BASIC- System zur Entgegennahme eines neuen Kommandos oder Anweisung bereit ist.
Sollten die Anweisungen in anderer Reihenfolge eingegeben worden sein, wird das LIST- Kommando eine Auflistung in aufsteigender Reihenfolge der Anweisungsnummern bewirken.

```
                        *40 PRINT A;" +";B;" =";C
Das Programm sei in     *20 LET C=A+B
dieser Reihenfolge ein- *30 PRINT
gegeben worden:         *10 INPUT A,B

                        *LIST
In der Auflistung sind    10 INPUT A,B
die Anweisungen in        20 LET C=A+B
richtiger Reihenfolge     30 PRINT
                          40 PRINT A;" +";B;" =";C
                        *
```

2.9. Starten eines Programms

Nachdem das Programm eingegeben und, nach Wunsch, mittels LIST kontrolliert ist, kann es mit dem RUN- Kommando ausgeführt werden. RUN wird anhand zweier Beispiel- Programme illustriert.

Das erste Programm hat eine Zeile

```
*100 PRINT "5 x 10 = ";5x10
```

Wenn es gestartet ist, wird das Ergebnis des Ausdrucks 5x10 ausgegeben:

```
*RUN
5 x 10 = 50
READY
*
```

Weil das Programm eine PRINT- Anweisung enthält, wird das Ergebnis während des Laufs des Programms ausgegeben.

Das zweite Beispiel-Programm addiert zwei Zahlen. Die Zahlen müssen vom Benutzer eingegeben werden:

```
*10 INPUT A,B
*20 LET C=A+B
*30 PRINT
*40 PRINT A;" +";B;" =";C
```

Die beiden Buchstaben hinter dem Wort INPUT, die durch Komma getrennt sind, bezeichnen Variablen, die je einen vom Benutzer am Bürocomputer eingegebenen Wert aufnehmen. Wenn das Programm gestartet ist, signalisiert der Interpreter durch ein Fragezeichen, daß Eingabe erwartet wird. Der Benutzer tippt die Werte hinter das Fragezeichen. Je zwei Werte werden durch Komma getrennt.
Die Anweisung LET C=A+B weist der Variable C links vom Gleichheitszeichen den Wert des Ausdrucks rechts vom Gleichheitszeichen zu. Der Ausdruck addiert die Werte der Variablen A und B. Das Ergebnis ist der Wert von C.

Wenn das Programm gestartet wird, gibt der Benutzer Eingabe- Werte ein und der Computer gibt das Ergebnis aus

```
*RUN
?1078,5.3
1078 + 5.3 = 1083.3
*
```

2 .10. Löschen eines Programms

Wenn ein Programm eingegeben wurde und gelaufen ist und dann nicht länger benötigt wird, kann es mit dem NEW-Kommando gelöscht werden. Die Eingabe von NEW löscht jedes vom Benutzer in der laufenden Sitzung bisher eingegebene Programm.
Das erste eingegebene Programm war 100 PRINT "5 x 10 ="; 5 x 10. Nachdem es ausgeführt wurde, sollte es vor der Eingabe des nächsten Programms gelöscht werden; sonst werden beide Programme beim nächsten RUN ausgeführt. Sie werden dann in der Ordnung ihrer Anweisungsnummern ausgeführt. Wenn z. B. beide Programme sich gegenwärtig im Benutzer- Arbeits- Bereich befinden, werden die Anweisungen mit den Nummern 10 bis 40 vor der Zeile 100 ausgeführt.

Beide Programme laufen bei Eingabe von RUN

```
*100 PRINT "5 x 10 =";5x10
*10 INPUT A,B
*20 LET C=A+B
*30 PRINT
*40 PRINT A;" +";B;"=";C
*RUN
?1078,5.3
1078 + 5.3 = 1083.3
5 x 10 = 50
```

Um verwirrende Ergebnisse zu vermeiden, kann ein eingegebenes Programm, das bereits gelaufen ist, mit NEW gelöscht werden:

```
                        *100 PRINT "5 x 10 =";5x10
Nach Eingabe und Lauf   *RUN
wird das Programm       *5 x 10 = 50
gelöscht:               *NEW
```

Der Benutzer- Arbeits- Bereich ist jetzt wieder frei und ein anderes Programm kann eingegeben werden.

```
                        *10 INPUT A,B
                        *20 LET C=A+B
Das zweite Programm     *30 PRINT A;" +";B;" =";C
wird eingegeben:        *RUN
                        ?343,275
                         343 + 275 = 618
```

Wenn dieses Programm nicht erneut laufen soll, kann es nun gelöscht werden und ein drittes Programm kann eingegeben werden.

2 .11. Dokumentation eines Programms

Bemerkungen, die zur Erklärung oder als Kommentar dienen, können in das Programm als REM- Anweisung eingefügt werden. Jede Bemerkung hinter REM wird in der Auflistung des Programms mit ausgegeben; sonst findet keine Beeinflussung der Programmausführung statt. Es können soviele REM- Anweisungen in das Programm eingefügt werden wie nötig.

```
Das Beispielprogramm    *5 REM...DIESES PROGRAMM ADDIERT
zur Addition von zwei   *7 REM 2 ZAHLEN
Zahlen wird mit mehre-  *15 REM...2 WERTE EINGABE
ren Bemerkungen ver-    *35 REM C ENTHAELT DIE SUMME
sehen:
```

Die Anweisungsnummern bestimmen die Position der Bemerkungen innerhalb des schon existierenden Programms. Eine Auflistung zeigt die Anordnung

Auflistung des Beispielprogramms einschließlich der Bemerkungen:

```
*LIST
  5 REM...DIESES PROGRAMM ADDIERT
  7 REM 2 ZAHLEN
 10 INPUT A,B
 15 REM... 2 WERTE EINGABE
 20 LET C=A+B
 30 PRINT
 35 REM C ENTHAELT DIE SUMME
 40 PRINT A;" +";B;" =";C
```

Wenn dieses Programm nun gestartet wird, wird es sich genauso verhalten wie vor Einfügung der Bemerkungen.
Kommentare können auch zusammen mit der Anweisung in einer Zeile stehen. Dies wird durch ein dem Text des Kommentars vorgestelltes "\" Zeichen erreicht. Zu beachten ist, daß bei Eingabe des Kommentars nach einer Anweisung zwei "\" Zeichen eingesetzt werden.
Zeichen hinter diesem Zeichen werden nicht als Teil der Anweisung behandelt, aber werden zusammen mit der Programm-Anweisung gespeichert.
Kommentare dieser Art im Anschluß an eine Anweisung können nicht in der selben Zeile mit einer DATA-Anweisung (Abschnitt 5.9.) benutzt werden.

Beispielprogramm mit Kommentaren hinter Anweisungen:

```
*10 INPUT A,B 2 ZAHLEN EINGABE
*20 LET C=A+B\ BERECHNE SUMME A+B
*30 PRINT\ ZEILENVORSCHUB
*40 PRINT A;" +";B;" =";C\DRUCKE
    SUMME
```

3. BASIC- Kommandos

Bisher haben wir die Kommandos LIST, RUN und NEW zur Manipulation einfacher Programme benutzt. Sowohl LIST als auch RUN haben Parameter und Funktionen, die noch nicht gezeigt wurden. Der volle Umfang der Kommandos, mit denen man den Lauf eines Programms kontrolliert, es editiert und es auf eine Diskette rettet ist:

RUN
STEP
CONTINUE

Die Editier- Kommandos:

LIST
NEW
DELETE
RENUMBER
SIZE
CLEAR

Diskettenbezogene Kommandos:

SAVE
ASAVE
GET
XEQ
APPEND
RSAVE

Kommandos werden direkt hinter das Bereitschaftszeichen "*" gesetzt und unmittelbar ausgeführt. Alle Kommandos können mit ihren ersten drei Buchstaben abgekürzt werden. Bei der Beschreibung der einzelnen Kommandos werden bestimmte Konventionen benutzt:

GROSS-BUCHSTABEN	Schlüsselworte, die genau so geschrieben werden müssen
klein-buchstaben	von Benutzer definierte Wörter
[ ]	umschließen optionale Begriffe
( )	umschließen vorgeschriebene Begriffe
|	trennt Alternativen, von denen eine gewählt werden muß
...	deutet mögliche Wiederholung des letzten Begriffs an

3.1. Programmausführungskommandos

Die Programmausführungskommandos unterstützen die Fehlersuche in einem Programm. Die Programmausführung kann entweder durch von-Hand-Eingriff oder unter Programmkontrolle unterbrochen werden. Variablen können beobachtet und/oder geändert werden, Programmteile können gezeigt oder geändert werden, danach die Ausführung fortgesetzt werden.

3.1.1. RUN

Das RUN- Kommando bewirkt die Ausführung eines SIOS-BASIC- Programms. Das Format ist

RUN[-zeilennummer]

Ist keine Zeilennummer angegeben, beginnt die Ausführung mit der ersten ausführbaren Anweisung. Wenn innerhalb des Programms ein STOP (Abschnitt 5.2.) ausgeführt wird, kann es durch Eingabe des CONTINUE- Kommandos fortgesetzt werden (Abschnitt 3.1.3.). Ist eine RUN- Zeilennummer angegeben, beginnt die Ausführung bei der ersten ausführbaren Anweisung bei oder hinter dieser Nummer. Der Startpunkt darf nicht innerhalb einer Funktions- Definition liegen.

3.1.2. XEQ

Das XEQ- Kommando lädt und startet ein SIOS- BASIC- Programm. Es ist äquivalent mit einem GET (Abschnitt 3.3.4.) gefolgt von einem RUN.

XEQ-programmname

Das Programm programmname wird in den Arbeitsbereich des Benutzers geladen und gestartet. (Siehe in Abschnitt 3.3. die Beschreibung der Konventionen für Dateinamen.)

Beispiel: *XEQ-BAGELS

Das Programm BAGELS wird in den Arbeitsbereich des Benutzers geladen und gestartet.

3 .1 .3 . Wiederaufnahme der Programmausführung durch CONTINUE, STEP, RUN

Wenn ein Programm durch STEP oder durch Betätigung der DEL- Taste unterbrochen ist, kann der Benutzer die Weiterausführung auf eine von mehreren Weisen veranlassen.

Format

CONTINUE
STEP
RUN-zeilennummer

Jedes dieser Kommandos bewirkt, daß die Programmausführung ohne jede Änderung einer Programmvariablen fortgesetzt wird.

Beschreibung

Das CONTINUE- Kommando bewirkt die Fortsetzung bei der nächsten auszuführenden Anweisung (bezogen auf den Zeitpunkt der Unterbrechung des Programms). Dieses Kommando kann stets gegeben werden, wenn das Programm im STOP- Zustand ist.
Das STEP- Kommando bewirkt die Fortsetzung des Programmlaufs bis zum Beginn der nächsten Anweisung auf der äußeren Ebene (d. h., einer, die nicht Teil einer Mehr- Zeilen- Funktion ist). Dieses Kommando kann benutzt werden, um schrittweise ein Programm zu durchlaufen. Dabei werden ganze Anweisungen der äußeren Ebene ohne weitere Unterteilung von Mehr- Zeilen- Funktionen ausgeführt, bevor das Programm wieder in den STOP- Zustand geht. STEP kann stets gegeben werden, wenn das Programm im STOP- Zustand ist.
"RUN-zeilennummer" bewirkt die Wiederaufnahme der Ausfüh-

rung bei der angegebenen Zeile. Dieses Kommando kann nicht gegeben werden zum Zweck der Fortsetzung mitten in einer Mehr- Zeilen- Funktion oder einem neuen DO-DOEND-Block, sollen nicht unerwünschte Seiteneffekte in Kauf genommen werden.
Wenn ein Programm unterbrochen ist, können die folgenden Kommandos gegeben werden, ohne daß dadurch die Fortsetzung des Programmlaufs unmöglich würde:

Jede im Tischrechner- Modus ausführbare Anweisung (Abschnitt 13.)

LIST
SAVE
ASAVE
SIZE

Die Programmausführung kann nach folgenden Kommandos nicht mehr fortgesetzt werden:

Programmänderung
RENUMBER
NEW
QUIT
GET
XEQ

3 .1 .4 . QUIT

Das QUIT- Kommando wird zum Verlassen des BASIC- Systems benutzt. Alle geöffneten Dateien werden geschlossen und der Benutzer- Arbeitsbereich gelöscht. Die Kontrolle geht wieder an SIOS über.

QUIT

3 .2 . Editierkommandos

Die Editier- Kommandos beziehen sich stets auf das aktu-

elle Programm, dasjenige Programm, das augenblicklich am Bürocomputer eingegeben oder bearbeitet wird.

3.2.1. LIST

Das LIST- Kommando listet vollständig oder teilweise das aktuelle Programm auf, es hat die Form

LIST [-bereich]

wobei bereich angibt, welche Anweisungen gezeigt werden sollen. Wird kein Bereich angegeben, wird das ganze Programm aufgelistet. Das Format und die Wirkung der Bereichsangabe sind folgende:

LIS-n	nur Anweisung n wird ausgegeben
LIS-n,	alle Anweisungen ab n bis zum Ende des Programms werden aufgelistet
LIS-,n	alle Anweisungen vom ersten bis zum angegebenen (einschließlich) werden aufgelistet
LIS-n,m	die Anweisungen ab n bis m (beide einschließlich) werden aufgelistet

Beispiele

```
*LIST
```

Das ganze aktuelle Programm wird am Bürocomputer aufgelistet.

```
*LIST-1,100
```

Die Anweisungen 1 bis 100 des aktuellen Programms werden aufgelistet.

Bemerkung:

Eine Auflistung kann durch Betätigung der DEL- Taste abgebrochen werden. Die Kontrolle kehrt ans BASIC- System zurück. Die Auflistung kann auch durch Druck der "?"- Taste unterbrochen werden. Der erneute Druck von "?" setzt

die Ausgabe fort.

3.2.2. NEW

Das NEW- Kommando löscht das aktuelle Programm, es hat die Form:

NEW
NEW-puffer

Wenn die Zahl der anzulegenden Puffer (0-15) nicht angegeben ist, werden zwei Puffer angelegt. Jeder Puffer umfaßt 512 Bytes. Die Puffer stellen zusätzliche Systembereiche dar, die bei der Ein- und Ausgabe mit Dateien benötigt werden.

Beispiel

```
*NEW-7
```

Das aktuelle Programm wird gelöscht, sieben Puffer werden angelegt und ein neues aktuelles Programm kann in den Benutzer- Arbeits- Bereich eingegeben werden. Das SIZE-Kommando (Abschnitt 3.2.5.) zeigt die Zahl der angelegten Puffer.

```
*SIZ:   AVAIL=9213 PROG=0 VAR=0 BUF=7
```

3.2.3. DELETE

Das DELETE- Kommando löscht ein oder mehrere angegebene Anweisungen, es hat die Form:

DELETE-bereich

wobei bereich unten beschrieben ist; die durch Parameter angegebenen Anweisungen werden aus dem Programm entfernt.

DEL-n	löscht Zeile n
DEL-,n	löscht alle Zeilen vom Anfang des Programms bis zur Zeile n (einschließlich)

DEL-n, löscht alle Zeilen ab Zeile n bis zum Ende des Programms

DEL-n,m löscht alle Zeilen ab Zeile n bis Zeile m (beide einschließlich)

Beispiel

*DEL-37,43

Alle Anweisungen ab 37 bis 43 einschließlich werden aus dem aktuellen Programm des Benutzers entfernt.

3 .2 .4 . RENUMBER

Voraussetzung für die Abarbeitung dieses Kommandos ist das Vorhandensein der Programmdatei RENUMBER.
Das RENUMBER- Kommando erlaubt dem Benutzer die Umnumerierung jeder Gruppe von Anweisungen im aktuellen Programm; es hat die Form:

RENUMBER-[neuerst [,delta[,alterst[,altletzt]]]]

alterst und altletzt geben den Bereich ursprünglicher Anweisungen an, der umnumeriert werden soll (Voreinstellung ist 1,9999). Wenn nur alterst angegeben ist, ist für altletzt 9999 voreingestellt. Der ersten dieser Anweisungen wird die Nummer neuerst (voreingestellt:10) zugewiesen. Jede folgende Anweisung erhält also eine Nummer, die um delta (voreingestellt:10) größer ist als die Vorgängerin. Jede Anweisung, die auf eine umnumerierte Anweisung verweist, wird sinngemäß geändert.

Beispiel

*RENUMBER

Die Anweisungen des aktuellen Programms werden in 10er-Schritten beginnend mit 10 umnumeriert.

*REN-3,7,50,250

Die Anweisungen mit den alten Nummern 50 bis 250 werden umnumeriert. Kleinste Nummer ist 3, Schrittweite ist 7.

3 .2 .5 . SIZE

Das SIZE- Kommando berichtet über den Zustand des aktuellen Programms.

Format

SIZE

Beispiel

```
*SIZ:AVAIL=9460 PROG=36 VAR=24 BUF=2
```

AVAIL zeigt die Zahl der verfügbaren Bytes an. PROG zeigt die Größe des durch das Programm belegten Speichers in Bytes an. VAR zeigt die Zahl der durch Programm- Variablen belegten Bytes an. BUF gibt die Zahl der angelegten Puffer an(siehe NEW, Abschnitt 3.2.2.).

3 .2 .6 . CLEAR- Kommando

Das CLEAR- Kommando bewirkt, daß alle Variablen undefiniert werden. Der belegte Speicherplatz wird wieder frei. Alle hängenden Funktionsaufrufe, GOSUBs und FORs werden zurückgesetzt, alle Dateien geschlossen. Die Form ist:

CLEAR

CLEAR gibt jeden während der Programmausführung belegten Speicherplatz wieder frei. Ein CLEAR wird automatisch ausgeführt, wenn RUN gegeben wird.

Beispiel

```
*SIZ:  AVAIL=9460 PROG=36 VAR=24 BUF=2
*CLEAR
*SIZ:  AVAIL=9520 PROG=0  VAR=0  BUF=2
```

Beispiele mit Editier- Kommandos:

Der Benutzer gibt ein Programm ein, macht einen Fehler in Zeile 30 und gibt die Zeile neu ein.

```
*10 INPUT A,B,C,D,E
*20 REM..EINGABE 5 WERTE
*30 LET R=(A+B)/5
*40 REM..S=MITTELWERT DER 5 EINGABEWERTE
*50 PRINT S
*30 LET S=(A+B+C+D+E)/5
```

LIST listet das Programm richtig auf:

```
*LIST
  10 INPUT A,B,C,D,E
  20 REM..EINGABE 5 WERTE
  30 LET S=(A+B+C+D+E)/5
  40 REM..S=MITTELWERT DER 5 EINGABEWERTE
  50 PRINT S
```

SIZE gibt die Länge in Bytes an:

```
*SIZ:  AVAIL=11648 PROG=125 VAR=0 BUF=2
```

Die Kommentarzeilen werden gelöscht und das Programm aufgelistet:

```
*DELETE-20
*DELETE-40
*LIST
  10 INPUT A,B,C,D,E
  30 LET S=(A+B+C+D+E)/5
  50 PRINT S
*SIZ:  AVAIL=11710 PROG=63 VAR=0 BUF=2
```

Als nächstes wird das Programm umnumeriert und erneut aufgelistet:

```
*RENUMBER
*LIST
  10 INPUT A,B,C,D,E
  20 LET S=(A+B+C+D+E)/5
  30 PRINT S
```

Das Programm wird gelöscht. Wenn nun LIST gegeben wird,

gibt es kein aktuelles Programm; der Computer bringt ein "*" und erwartet weitere Eingabe:

```
* NEW
* LIST
*
```

3.3. Diskettenbezogene Kommandos

Wenn ein Programm zu einem späteren Zeitpunkt wiederverwendet werden soll, kann es auf Diskette zwischengespeichert werden. Dazu muß es im SAVE-, ASAVE- oder KSAVE Kommando einen Namen bekommen.
Die Abarbeitung des SAVE- und ASAVE-Kommandos bewirkt eine automatische Anlage der Programmdatei. Existiert eine Datei mit dem im Komando angegebenen Namen wird eine Fehlermeldung ausgegeben. Der Programmname und der Dateiname können aus bis zu 17 signifikanten Stellen bestehen.
Der Programmname wird benutzt, um die Kommandos GET (Holen) und APPEND (Anhängen) auf das Programm anwenden zu können.

3.3.1. SAVE

Das SAVE- Kommando speichert eine Kopie des aktuellen Programms auf die Diskette, das Format ist

SAVE-programmname

In der entsprechenden Datei wir eine Programmkopie in compilierterer Form abgelegt. Diese Form ist kompakter als die Form des Quellprogramms und erlaubt ein schelleres Rückretten. Das Programm kann jedoch nur in Quellform vernünftig gelesen werden. Das ASAVE- Kommando (Abschnitt 3.3.2.) ergibt eine gerettete Version des BASIC- Programms in Quellform.

Beispiel

```
* SAVE-BEISPX
```

Der Name BEISPX wird einer Kopie des aktuellen Programms zugewiesen. In einer Datei mit gleichem Namen wird das Programm auf Diskette gerettet.

3.3.2. ASAVE

Das ASAVE-Kommando speichert eine Kopie des aktuellen Programms auf die Diskette. Das Programm wird in der Quelldarstellung abgelegt.

ASAVE-programmname

Das Programm wird normalerweise ohne Zeileneinrückung gerettet. Ein Aufruf SYSTEM "ASINON" (Abschnitt 5.12.) bewirkt, daß das Programm mit Einrückung der Zeilen gerettet wird.

Beispiel:

*ASA-PROGNAM

Der Name PROGNAM wird der Kopie des aktuellen Programms zugewiesen, die auf die Diskette gerettet wird.

3.3.3. RSAVE

Das RSAVE-Kommando arbeitet wie das ASAVE-Kommando, nur mit dem Unterschied, daß keine Kontrolle auf Existenz der Programmdatei gleichen Namens erfolgt. Damit kann der Inhalt einer Programmdatei gleichen Namens überschrieben werden.

3.3.4. GET

Das GET-Kommando lädt ein angegebenes BASIC-Programm in den Arbeitsbereich des Benutzers. Die Form ist:

GET-programmname

wobei programmname der Name des Programms ist, das das aktuelle Programm ersetzen soll.

Beispiel:

*GET- SINUS

SINUS ist ein Programm auf der Diskette. Ab jetzt steht es im Arbeitsbereich des Anwenders zur Verfügung, wo es jedes frühere ersetzt.

3.3.5. XEQ

Das XEQ-Kommando lädt und startet ein BASIC-Programm. Es ist äquivalent mit der Kommandofolge GET, RUN. Einzelheiten im Abschnitt 3.1.2.

3.3.6. APPEND

Das APPEND-Kommando hängt ein angegebenes Programm an das aktuelle Programm an. Es hat die Form:

APPEND-programmname

Das Programm programmname wird an das Ende des aktuellen Programms angehängt. Nur Programme, die mittels ASAVE gerettet worden sind, können angehängt werden Programme, die in pseudokompilierter Form (siehe SAVE-Kommando, Abschnitt 3.3.1.) gerettet wurden, können nicht angehängt werden. Zeilennummern, die im Arbeitsbereich bereits existieren, werden durch die entsprechenden aus der APPEND-Datei ersetzt.

Beispiel

```
*APPEND-BEISP
```

BEISP ist ein mit ASAVE gerettetes Programm auf der Diskette. Es wird an das aktuelle Programm im Benutzer- Arbeitsbereich angehängt.

Beispiele zur Verwendung der Disketten-Kommandos

Ein Programm wird eingegeben und auf Diskette gerettet. Das Programm wird danach gelöscht.

```
100 INPUT A,B,C,D,E
120 LET S= (A+B+C+D+E)/5
130 PRINT S
*ASA - AVERAGE
*NEW
```

Ein zweites Programm wird eingegeben und gerettet. Das erste Programm wird dann an dieses Programm angehängt und dies ergibt eines drittes Programm. Dieses wird ebenfalls gerettet:

```
* 10 INPUT R
* 20 P=3.14
* 30 A=PxR^2
* 40 PRINT A
* SAVE-AREA
* APPEND-AVERAGE
* SAVE-CALC
```

Jedes dieser Programme kann nun als aktuelles Programm mittels GET zurückgebracht werden. Zur Anschauung wird jedes wiedergeholt und dann aufgelistet:

```
*GET-AVERAGE
*LIST
  100 INPUT A,B,C,D,E
  120 LET S=(A+B+C+D+E)/5
  130 PRINT S

*GET-AREA
*LIST
   10 INPUT R
   20 LET P=3.14
   30 LET A=PxR^2
   40 PRINT A

*GET-CALC
*LIST
   10 INPUT R
   20 LET P=3.14
   30 LET A=PxR^2
   40 PRINT
  100 INPUT A,B,C,D,E
  120 LET S=(A+B+C+D+E)/5
  130 PRINT S
```

4 . Die Grundelemente der Sprache BASIC

BASIC ist eine zeilenorientierte Sprache. Ein Programm ist eine Folge von Anweisungen und jede Anweisung wird durch ein Schlüsselwort identifiziert. Jede Zeile ist mit einer nur einmal vorkommenden Zeilennummer versehen, die als Marke für die in der jeweiligen Zeile enthaltenen Anweisung dient.
Programmzeilen werden in sequentieller Reihenfolge bearbeitet. Die Zeilen sind durch die Zeilennummern geordnet.

Beispiel

```
*10 INPUT A,B
*20 LET C=A+B
*30 PRINT A;" +";B;" =";C
```

Für die vollständige Angabe der Anweisungen werden Grundelemente verwendet. Dazu gehören Konstanten, Variablen, Funktionen, Operatoren und Ausdrücke, die im nachfolgenden beschrieben werden.

4 .1 . Die Grundsymbole von SIOS- BASIC

Die Grundsymbole Buchstabe, Ziffer, Sonderzeichen und Schlüsselwort sind die kleinsten Sprachelemente. Aus ihnen wird entsprechend den syntaktischen Regeln das Programm aufgebaut.

1. Buchstaben =A/B/C/D/E/F/G/H/I/J/K/L/M/N/O/P/Q/R/S/T U/V/W/X/Y/Z/a/b/c/d/e/f/g/h/i/j/k/l/m/n o/p/q/r/s/t/u/v/w/x/y/z

2. Ziffern =0/1/2/3/4/5/6/7/8/9

3. Sonderzeichen =!/"/#/¤/%/&/'/(/)/*/+/-/,/./@/:/;/</=/^ ?/>/_/„/@/[/\/]/`/{/|/}/~

4. Schlüsselwort =CALL/CHAIN/CLOSE/COM/DATA/DEF/DIM/DO/ DOEND/ELSE/END/ERASE/FILE/FNEND/FOR/ NEXT/GOTO/GOSUB/IF/THEN/INPUT/LET/LINPUT/ ON/RESTORE/PRINT/USING/RANDOMIZE/READ/ REM/RETURN/SPACE/STOP/SYSTEM/TRUNCATE/ WRITE

4 .2 . Konstanten

Eine Konstante ist entweder eine Zahl oder eine Zeichenkette.

4 .2 .1 . Zahlkonstanten

Eine Zahl- Konstante ist eine positive oder negative Dezimalzahl einschließlich der Null. Die Zahl- Konstanten sind auf 13 Dezimalstellen genau. Sie können als Integer (ganzzahlig), Fest- Punkt- Zahl oder Gleitkomma- Zahl geschrieben werden. Eine Beschreibung der Gleitkommadarstellungen findet sich in Abschnitt 6.2.2.
Ganze Zahlen sind Folgen von Ziffern ohne Dezimalpunkt.

Beispiele:

```
 1234567890123
-1234567890123
 0
-56789
```

Gleitkomma- Zahlen sind Zahlen gefolgt von einem Buchstaben E und einer ganzen Zahl, gegebenenfalls mit Vorzeichen. In der Gleitkomma- Schreibweise ist die Zahl vor dem E eine Größe, die noch mit einer 10er- Potenz multipliziert wird. Die ganze Zahl hinter dem E ist der Exponent (Hochzahl), d. h. sie gibt die 10er- Potenz an, mit der die Größe multipliziert wird.
Der Exponent einer Gleitkommazahl wird benutzt, den Dezimalpunkt zu setzen; ohne diese Schreibweise wäre die Beschreibung einer sehr großen oder sehr kleinen Zahl umständlich.

```
1E+35 = 100000000000000000000000000000000000
1E-35 = .00000000000000000000000000000000000
```

Beispiele für Gleitkommazahlen:

```
 1E+23    = 1 x 10^23 = 100000000000000000000000
 1.0E23     (wie oben)
 .001E26    (wie oben)
 1.02E+7  = 1.02 x 10^7 = 10200000
 1.02E-7  = .000000102
```

Intern werden alle diese Zahlen als Gleitkommazahlen mit

einer Größe zwischen 1E-128 und 1E+127 dargestellt.

Gleitkommazahlen

```
 12345.789012E+20
-1234567890123E+5
 123456.0789E-5
```

4 .2 .2 . Zeichenketten

Eine Zeichenkette besteht aus einer Folge von Zeichen aus dem KOI-7Bit- Zeichensatz, eingeschlossen in Anführungszeichen ("). Die Anführungszeichen selbst und das Zeichen "<" sind die einzigen Zeichen, die als nicht zur Zeichenkette gehörig betrachtet werden. Leerzeichen haben in der Zeichenkette Bedeutung.

```
"ABC"
"!WAS FUER EIN TAG!!"
" X Y  Z  "
" "     (leere Zeichenkette, Zeichenkette der Länge 0)
" "     (Zeichenkette, bestehend aus zwei Leerzeichen)
```

4 .3 . Variablen

Eine Variable ist ein Name, dem ein Wert zugeordnet ist. Dieser Wert kann während der Programmausführung geändert werden. Das Auftreten einer Variablen ist mit dem Auftreten des jeweiligen Wertes in seiner Wirkung gleichzusetzen. Variablen haben entweder Zahlwert oder Zeichenkettenwert.
Relle Variablen bestehen aus einem einzigen Buchstaben (von A bis Z) oder einem Buchstaben und einer direkt folgenden Ziffer (von 0 bis 9).

```
A        A0
```

P P5
X X9

Eine Variable dieses Typs enthält stets einen Zahlwert, der intern als relle Gleitkommazahl dargestellt ist. Variablen können auch Werte haben, die intern als 16-Bit-Ganzzahl dargestellt sind. Namen solcher Variablen sind ähnlich wie die obigen, mit der Ausnahme, daß ihre Namen ein angehängtes Prozentzeichen "%" haben:

A% B5%
X% X3%

Variablen können als Wert eine Zeichenkette besitzen. Dieser Typ von Variablen ist gekennzeichnet durch einen Variablennamen, der ähnlich wie die obigen gebildet werden, jedoch auf einem Währungszeichen "¤" enden:

A¤ AO¤
P¤ P5¤

Der Wert einer Zeichenketten- Variablen ist immer eine Zeichenkette, möglicherweise eine leere . Zeichenketten-Variablen können nicht benutzt werden, ohne daß sie zuvor mittels einer DIM- Anweisung deklariert wurden (siehe Abschnitt 8.3.).
Wenn eine Variable ein Feld (siehe Felder, Abschnitt 7.) benennt, dann kann sie indiziert werden. Wenn eine Variable indiziert ist, steht hinter dem Variablennamen die Indexliste, bestehend aus einem oder zwei Indizes, eingeschlossen in Klammern. Falls es zwei Indizes gibt, werden diese durch Komma getrennt. Ein Index kann eine ganzzahlige Konstante sein oder eine Variable oder ein beliebiger Ausdruck, der auf einen ganzzahligen Wert gerundet wird:

A(1) AO(N%,M%)
P(1,1) P5%(Q5,N/2)
x(N+1) X9(10,10)

Eine einfache (nicht indizierte Variable) und eine indi-

zierte Variable, beide vom Zahltyp (keine Zeichenketten) können denselben Namen haben, ohne daß dies einen Zusammenhang impliziert. Die Variable A hat nichts mit der Variablen A(1,1) zu tun.
Einfache Zahlen- Variablen können benutzt werden, ohne deklariert zu sein. Indizierte Zahlen- Variablen müssen durch eine DIM- Anweisung deklariert werden (siehe Abschnitt 7.1.), wenn die Dimension des Feldes größer ist als 10 Zeilen oder 10 Zeilen und 10 Spalten. Der erste Index bezeichnet stets die Zeilennummer, der zweite die Spaltennummer. Der gerundete Wert des Index muß zwischen 1 und der in der Deklaration angegebenen oberen Grenze für den Zeilen- bzw. Spaltenindex liegen.
Zeichenketten- Felder unterscheiden sich von Zahlen- Feldern dadurch, daß sie nur eine Dimension haben, folglich nur einmal indiziert werden. Auch dürfen ein Zeichenketten- Feld und eine einfache Zeichenketten- Variable nicht den gleichen Namen haben (siehe Zeichenketten- Felder in Abschnitt 8.). Beispiele für indizierte Namen von Zeichenketten- Feldern sind:

A$(1) AO$(N) B5$(Z%) X$(Y-3)

4 .4 . Funktionen

Eine Funktion bezeichnet eine Operation, die unter Verwendung von einem oder mehreren Parameter- Werten einen einzelnen Wert als Ergebnis produziert. Eine Zahlwert-Funktion wird identifiziert durch einen Mehr- Buchstaben-Namen (oder einen Mehr- Buchstaben- Namen plus einem Prozentzeichen ("%")). Es folgen einer oder mehr Parameter in Klammern. Wenn es mehr als einen Parameter gibt, sind diese durch Kommas getrennt. Zahl und Typ der Parameter hängen von der jeweiligen Funktion ab. Die formalen Parameter in der Funktions- Definition werden durch die aktuellen Parameter ersetzt, wenn die Funktion benutzt wird.

Da eine Funktion einen einzigen Wert als Ergebnis liefert, kann sie überall in einem Ausdruck wie eine Konstante oder eine Variable verwendet werden. Bei Verwendung der Funktion wird der Funktions- Name gefolgt von der geklammerten Liste der Aktual- Parameter in einem Ausdruck eingesetzt. Dies nennt man Funktions- Aufruf. Der Ergebnis-Wert wird bei der Auswertung des Ausdrucks benutzt.

Beispiele gewöhnlicher Funktionen:

INT(X) wobei X ein Zahlwert- Ausdruck ist. Bei Aufruf wird die größte ganze Zahl, die kleiner oder gleich X ist, zurückgegeben.
(z.B.: INT (8.35)=8)

SGN(X) wobei X ein Zahlwert- Ausdruck ist. Bei Aufruf wird 1 bei X>0,
0 bei X=0 oder
-1 bei X<0 zurückgegeben.
(z.B.: SGN (4*-3)=-1)

SIOS- BASIC besitzt viele eingebaute Funktionen, die verbreitete Operationen ausführen wie: Sinus, Quadratwurzel oder Absolutwert einer Zahl. Die verfügbaren Zahlwert-Funktionen sind im Anhang aufgeführt und in Abschnitt 6.8. beschrieben. Zusätzlich kann der Benutzer neue Funktionen definieren. Wie man Funktionen definiert und aufruft, ist in Abschnitt 9., Benutzerdefinierte Funktionen,beschrieben.
Die bisher beschriebenen Funktionen sind Zahlwert- Funktionen, die als Ergebnis einen Zahlwert liefern. Es gibt auch Funktionen, die als Ergebnis eine Zeichenkette liefern. Diese werden durch einen Mehr- Buchstaben- Namen mit angeschlossenem $ bezeichnet. Zeichenketten- Funktionen werden zusammen mit Benutzer- definierten Funktionen in Abschnitt 9. beschrieben. Verfügbare eingebaute Funktionen sind im Anhang aufgeführt und in Abschnitt 8.5. beschrieben.

4.5. Operatoren

Ein Operator führt eine mathematische oder logische Operation auf einen oder zwei Werte aus und liefert einen einzelnen Wert. Generell wird ein Operator zwischen zwei Werte gesetzt. Es gibt aber auch einstellige Operatoren, die vor einen einzelnen Wert gesetzt werden. Zum Beispiel ist das Minus- Zeichen in A-B ein zweistelliger Operator und bezeichnet die Subtraktion; der Ausdruck liefert die Differenz (positiv oder negativ) beider Werte. Das Minus-Zeichen in -A ist ein einstelliger Operator und kehrt das Vorzeichen von A um.

Die Kombination von einem oder mehreren Operanden mit einem Operator bildet einen Ausdruck. Die in einem Ausdruck auftretenden Operanden können Konstanten, Variablen, Funktionen oder selbst wieder Ausdrücke sein.

Operatoren können in Typen eingeteilt werden, je nach Art der Operation. Die Haupt- Typen sind: arithmetische, Vergleichs- und logische (Boolsche) Operatoren.

Die arithmetischen Operatoren sind:

+	Addition (einstellig: positiv) A+B oder +A
-	Subtraktion (einstellig: negativ) A-B oder -A
*	Multiplikation A * B
/	Division A / B
^	Exponentiation A ^ B

In einem Ausdruck bewirken die arithmetischen Operatoren die Errechnung eines einzelnen Zahlwertes.

Die Vergleichsoperatoren sind:

=	Gleich	A=B
<	Kleiner als	A<B
>	Größer als	A>B
<=	Kleiner oder gleich	A<=B
>=	Größer oder gleich	A>=B
<>	Ungleich	A<>B

Bei Auswertung von Vergleichsoperatoren in einem Ausdruck ergibt sich der Wert 1, wenn der angegebene Vergleich stimmt, bzw. der Wert 0, wenn der Vergleich nicht stimmt. Z.B. wird für A=B der Wert 1 berechnet, wenn A und B gleich sind; sonst ergibt sich 0.

Logische oder Boolsche Operatoren sind:

&	Logisches "und"	A&B
!	Logisches "oder"	A!B
-	Logisches Komplement	-A

Wie bei Vergleichsoperatoren bringt die Auswertung eines Ausdrucks mit logischen Operatoren den Wert 1, wenn der Ausdruck wahr ist, oder den Wert 0, wenn der Ausdruck nicht wahr ist.

Logische Operatoren werden wie folgt ausgewertet:

A&B = 1 (wahr) wenn A<>0 und B<>0
0 (falsch) wenn A=0 oder B=0

A!B = 1 (wahr) wenn A<>0 oder B<>0
0 (falsch) wenn A=0 und B=0

-A = 1 (wahr) wenn A=0
0 (falsch) wenn A<>0

Es ist ein Zeichenketten- Operator verfügbar, der zwei Zeichenketten zu einer verknüpft.

+	Verkettung	A¤ + B¤

Die Werte von A¤ und B¤ werden zur Bildung einer einzelnen Zeichenkette verwendet. Die Zeichen in B¤ werden direkt an die von A¤ "angehängt". Es sei A¤ ="ABC" und B¤ ="DEF"; dann ergibt sich für A¤ + B¤ der Wert "ABCDEF" (siehe Zeichenketten, Abschnitt 8.).

4 .6 . Ausdrücke und deren Auswertung

Ein Ausdruck verbindet Konstanten, Variablen oder Funk-

tionen durch Operatoren zu einer geordneten Folge. Bei Auswertung muß ein Ausdruck einen Wert ergeben. Zum Beispiel wird ein Ausdruck, der bei Auswertung einen ganzzahligen Wert (Integer-Wert) ergibt, Integer- Ausdruck genannt .
Konstanten, Variablen und Funktionen repräsentieren Werte; Operatoren geben an, welcher Typ von Operationen auf diesen Werten ausgeführt werden soll. Die Abschnitte 6. und 7. beschreiben den numerischen bzw. den Zeichenketten (string-) Typ im einzelnen.

Einige Beispiele für Ausdrücke:

(A*3)-(B+10)	A und B sind Variablen, denen zuvor ein Wert zugewiesen worden sein muß. 3 und 10 sind Konstante. Die zuerst auszuwertenden Teile des Ausdrucks sind geklammert.
	Es sei A=6 und B=4. Dann handelt es sich um einen Integer- Ausdruck mit dem Wert 4.
(X*(Y-2))+Z	Den Variablen X,Y und Z müssen zuvor Werte zugewiesen worden sein. *, + und - sind die Multiplikations-, Additions- bzw. Subtraktions- Operatoren. Der von der innersten Klammer zusammengeschlossene Teil wird zuerst ausgewertet.
	Wenn X=7, Y=4 und Z=3 ist, dann ist der Wert dieses Integer- Ausdrucks 17.

Die Auswertung eines Ausdrucks geschieht durch Ersetzung jeder Variablen durch ihren Wert, Auswertung jedes Funktions- Aufrufs und Durchführung der durch die Operatoren bezeichneten Operationen. Die Reihenfolge der Ausführung der Operationen wird durch die Rangordnung der Operatoren:

Höchste Priorität

einstelliges +, einstelliges -,
^
*,/
zweistelliges +, zweistelliges -
Vergleiche (=,<,>, <=, >=,<>)
& !

Niedrigste Priorität

bestimmt.
Die Operatoren der höchsten Stufe werden zuerst durchgeführt. Dann folgen die anderen in der oben angegebenen Prioritätsfolge. Bei zwei Operatoren auf derselben Stufe wird von links nach rechts gearbeitet. Klammern werden zur Aufhebung dieser Ordnung verwendet. Operationen, die geklammert sind, werden vor solchen außerhalb der Klammern durchgeführt. Bei Schachtelung von Klammern wird bei dem von der innersten Klammer umschlossenen Ausdruck begonnen.

Zum Beispiel:

5 + 6 * 7 wird berechnet nach: 5 +(6 * 7) = 47

7/14 * 2/5 wird berechnet nach: ((7/14) *2)/5 = .2

Es sei A=1, B=2, C=3, D=3.14, E=0

dann ergibt:		
	A+B*C	A+(B*C)=7
	A*B+C	(A*B)+C=5
	A+B-C	(A+B)-C=0
	(A+B)*C	(A+B)*C=9

Bei einem Vergleich bestimmt der Vergleichsoperator, ob der Vergleich als Ergebnis 1 (wahr) oder 0 (falsch) liefert. Wenn A,B und C die oben angegebenen Werte haben, dann ergibt

(A*B)<(A-C/3) den Wert 0 (falsch) denn:
A*B=2 ist nicht kleiner als A-C/3=0.

In einem logischen Ausdruck werden zuerst die anderen Operatoren ausgewertet, so daß man falsch- und wahr-Werte erhält. Die logischen Operatoren bestimmen dann, ob der ganze Ausdruck falsch (0) oder wahr (1) ist. Für das Beispiel von oben gilt dann:

E&A-C/3	ergibt 0(falsch), weil beide Seiten der Und- Verknüpfung 0 sind (E=0, A-C/3=0).
A+B&A≠B	ergibt 1 (wahr), weil beide Seiten ungleich 0 sind (A+B=3, A≠B=2).
A=B!C=SIN(D)	ergibt 0 (falsch), weil beide Seiten gleich 0 sind (A=B=0, C=SIN(D)=0).
A!E	ergibt 1 (wahr), weil eine Seite ungleich 0 ist (A=1, E=0).
¬E	ergibt 1 (wahr), weil E=0 ist.

Bezüglich Regeln über Auswertung von Ausdrücken mit Zeichenketten wird auf Abschnitt 8.6., Vergleich von Zeichenketten, verwiesen.

5. Anweisungen

Hier werden Anweisungen beschrieben, die zum Schreiben eines Programms unbedingt notwendig sind. Eine allgemeine Beschreibung von Anweisungen wurde in Abschnitt 2.5.2. gegeben. Man sollte sich ins Gedächtnis zurückrufen, daß jeder Anweisung eine Anweisungsnummer vorstehen muß und daß jede Anweisung durch Drücken der ETI Taste abgeschlossen wird. Diese Anweisungen werden nicht vor dem Start des Programms durch das RUN Kommando ausgeführt. Einige Anweisungen können auch unmittelbar ausgeführt werden und sind für Text und Fehlersuche nützlich (siehe Abschnitt 13.).

5.1. Zuweisungsanweisung

Diese Anweisung weist einer oder mehreren Variablen einen Wert zu. Der Wert kann in Form eines Ausdrucks, einer Konstanten, einer Zeichenkette oder einer anderen Variablen desselben Typs vorgegeben sein.

Format

Wenn der Wert eines Ausdrucks einer einzelnen Variablen zugewiesen wird, hat eine solche Anweisung eine der beiden Formen:

```
variable=ausdruck
LET variable=ausdruck
```

Mehrere Zuweisungen können in eine Anweisung gepackt werden, wenn sie durch Komma getrennt werden:

```
variable=ausdruck,...,variable=ausdruck
LET variable=ausdruck,...,variable=ausdruck
```

Das Wort LET ist ein nicht obligater Teil der Zuweisungs-Anweisung. Es kann entfallen.

Beschreibung

In dieser Anweisung ist das Gleichheitszeichen ein Zuweisung-Operator. Es deutet keine Gleichheit an, sondern ist das Signal, daß der Wert auf der rechten Seite des Zuweisungsoperators der Variablen links davon zugewiesen wird. Wenn die Variable, der ein Wert zugewiesen werden soll, indiziert ist, werden zuerst die Werte der Indizes berechnet. Die Berechnung der Indizes geschieht von links nach rechts; dann wird der Ausdruck ausgewertet und das Ergebnis wird zur Variablen übertragen.

Beispiele:

```
10 LET Z1=34.567
20 Z1=34.567
```

Der Variablen Z1 wird der Wert 34.567 zugewiesen. Die Anweisungen 10 und 20 haben denselben Effekt.

```
50 N=0
60 LET N=N+1
70 LET A(N)=N
```

Die Anweisungen 50 bis 70 setzen das Feldelement A(1) auf 1. Durch Wiederholung der Anweisungen 60 und 70, würde jedes Feldelement auf den Wert seines Index gesetzt.

```
 80 A=10.5,B=7.5
 90 B$="ABC",C$=B$
100 D%=5,E1%=10
```

Die reelle Variable A wird auf 10.5, dann B auf 7.5 gesetzt. Der Zeichenketten-Variablen B$ wird der Wert "ABC", dann C$ der Wert von B$ ("ABC") zugewiesen. Die Integer-Variable D% wird auf 5, dann E1% auf 10 gesetzt. Zeichenketten und Zeichenketten-Zuweisung werden in Abschnitt 8 behandelt.

5.2. END/STOP Anweisung

Die END- und STOP-Anweisungen werden zur Beendigung eines Programmlaufs benutzt. Es können beide benutzt werden, sie sind jedoch beide nicht erforderlich. Es wird angenommen, daß ein END hinter der letzten Zeile eines aktuellen Programms steht.

Format

```
END
STOP
```

Die END-Anweisung besteht aus dem Wort END, die STOP-Anweisung aus dem Wort STOP.

Beschreibung

Sowohl END als auch STOP beenden die Ausführung eines Programms. END hat eine andere Funktion als STOP; es bewirkt, daß alle Dateien geschlossen werden und die Meldung "READY" erscheint. STOP bewirkt die Ausgabe der Meldung "STOP AT nnnn", wobei nnnn die Anweisungsnummer der betreffenden STOP-Anweisung ist. Nach einem STOP kann die Programmausführung wieder aufgenommen werden (siehe Abschnitt 3.1.3).

Beispiele

Diese drei Programme haben die gleiche Wirkung:

```
   10 DIM A$[5], B$[15], C$[15]
   20 LET A$="HELLO", B$="THERE"
   30 C$= A$+" "+B$
   40 PRINT C$
*RUN
 HELLO THERE
 READY
```

```
   10 DIM A$[5], B$[15], C$[15]
   20 LET A$="HELLO", B$="THERE"
   30 C$= A$+" "+B$
   40 PRINT C$
   50 END
*RUN
 HELLO THERE
 READY
```

```
   10 DIM A$[15], B$[15], C$[15]
   20 LET A$="HELLO", B$="THERE"
   30 C$=A$+" "+B$
   40 PRINT C$
   50 STOP
*RUN
 HELLO THERE
 STOP AT   50
```

Wenn das Programm eine direkte Folge von Anweisungen ist und die letzte Anweisung des Programms auch die letzte ist, die tatsächlich ausgeführt wird, können END oder STOP entfallen. Dann erscheint zwar wie bei END die Meldung "READY", offene Dateien bleiben jedoch offen. END und STOP erfüllen einen Zweck, wenn die Abarbeitungsfolge nicht direkt ist und die letzte Anweisung des Programms nicht als letzte ausgeführt wird:

```
100 INPUT X
110 PRINT
120 GOSUB 140
130 END
140 IF X>0 THEN PRINT "X > 0"
150 ELSE PRINT "X <60> = 0"
160 RETURN
*RUN
 ?-356
X<= 0
```

Das Unterprogramm bei Zeile 140 folgt auf die END-Anweisung.

5.3. Schleifenanweisung: FOR...NEXT

Die Schleifen-Anweisungen FOR und NEXT ermöglichen die wiederholte Ausführung einer Gruppe von Anweisungen. Die FOR-Anweisung steht vor den zu wiederholenden Anweisungen, die NEXT Anweisung direkt dahinter. Die Zahl der Wiederholungen wird durch den Wert einer einfachen Zahlwert-Variablen bestimmt, die in der FOR-Anweisung angegeben ist.

Format

FOR variable=ausdruck TO ausdruck

FOR variable=ausdruck TO ausdruck STEP ausdruck

Die Variable ist entweder eine reelle oder eine ganzzahlige Variable. Sie wird zuerst auf denjenigen Wert gesetzt, der sich aus dem Ausdruck direkt hinter dem Gleichheitszeichen ergibt. Wenn der Wert der Variablen den Wert des Ausdrucks hinter dem TO überschreitet, wird die Schleife abgebrochen. Falls STEP angegeben ist, wird der Wert der Variablen nach jeder Wiederholung der Anweisungsgruppe um den Wert des hinter STEP angegebenen Ausdrucks erhöht. Dieser Wert kann positiv oder negativ sein; er sollte nicht 0 sein. Wenn STEP nicht angegeben ist, wird die Variable jeweils um 1 erhöht.

Die NEXT Anweisung schließt die Schleife ab:

NEXT variable

Die Variable hinter NEXT muß dieselbe sein wie die im zugehörigen FOR angegebene.

Beschreibung

Wenn FOR ausgeführt wird, wird der Variablen der Ausgangswert zugewiesen, der sich aus dem Ausdruck hinter dem Gleichheitszeichen ergibt. Ferner werden der End- und gegebenenfalls der Schrittwert berechnet. Dann werden folgende Schritte durchgeführt:

1. Der Wert der FOR-Variablen wird mit dem Endwert verglichen. Wenn jener größer ist als der Endwert (oder kleiner als der Endwert im Falle eines negativen STEP-Wertes), wird an die erste Anweisung hinter NEXT gesprungen. Sonst wird mit der Anweisung direkt hinter der FOR Anweisung fortgefahren.

2. Alle Anweisungen zwischen FOR und NEXT werden ausgeführt.

3. Die FOR-Variable wird um 1 erhöht, bzw. um den STEP-Wert, wenn ein solcher angegeben wurde.

4. Weiter bei Schritt 1.

Jedesmal, wenn eine FOR-Schleife begonnen wird, prüft das BASIC-System, ob das Programm sich nicht bereits in der Abarbeitung einer FOR-Schleife mit derselben FOR-Variablen befindet. Wenn das der Fall ist, werden alle aktiven Schleifen innerhalb und einschließlich der älteren Schleife mit der gleichen Variablen deaktiviert und weiter vorgegangen wie oben beschrieben.

Der Benutzer sollte Anweisungen innerhalb einer FOR-Schleife nicht anders ausführen als durch die FOR-Anweisung. Mitten in eine Schleife zu springen, kann zu unvorhersehbaren Ergebnissen führen, da die Schleifenparameter dann nicht richtig gesetzt sind.
FOR-Schleifen können geschachtelt werden. Eine FOR-Schleife ist dann vollständig in einer anderen enthalten. Sie dürfen sich nicht überlappen.

Beispiele

Bei jedem Durchlauf der FOR-Schleife wird ein kleinerer Bruch ausgegeben.

```
*10 FOR A=1 TO 16
*20   PRINT 1/(10^A)
*30 NEXT A
*RUN
  .1
  .01
  .001
  .0001
  .00001
  .000001
  .0000001
  .00000001
  .000000001
  .0000000001
  .00000000001
  .000000000001
  .0000000000001
  1.000000000000E-014
  1.000000000000E-015
  1.000000000000E-016
```

Die folgende FOR-Schleife wird sechsmal ausgeführt, wobei der Wert von X jedesmal um 1 erhöht wird:

```
  10 FOR X=0 TO -5 STEP -1
  20  PRINT X
  30 NEXT X
*RUN
 0
-1
-2
-3
-4
-5
```

Den ersten X Elementen des Feldes P(N) werden Werte zugewiesen. Wenn N=X, wird die Schleife beendet. In unserem Fall wird der Wert von X eingelesen:

```
*10 INPUT X
*20 PRINT
*30 FOR N=1 TO X
*40   LET P (N)=Nx10
*50   PRINT  P(N)
*60 NEXT N
*RUN
 ?6
  10
  20
  30
  40
  50
  60
```

Die folgenden Beispiele zeigen erlaubte und unerlaubte Schachtelung. Beim zweiten Beispiel wird eine Fehlermeldung gebracht, beim Versuch, das Programm zu starten:

```
10 REM..THIS EXAMPLE IS LEGAL
20 DIM Y[7,16]
30 FOR A=1 TO 7 STEP 2
40 FOR B=1 TO 16 STEP 2
50    LET Y(A,B)=-1
60  NEXT B
70 NEXT A
```

```
10 REM..THIS EXAMPLE IS ILLEGAL
20 DIM Y[7,16]
30 FOR A=1 TO 7 STEP 2
40   FOR B=1 TO 16 STEP 2
50    LET Y(A,B)=-1
60  NEXT A
70 NEXT B
*RUN
ERR:60 AT  70
```

5.4. GOTO/ON...GOTO Anweisungen

GOTO und ON...GOTO heben die normale Abarbeitungsreihenfolge der Anweisungen durch Sprung auf eine angegebene Anweisung auf. Die Anweisung, zu der gesprungen wird, muß im aktuellen Programm existieren.

Format

GOTO anweisungsnummer

ON ganzzahliger ausdruck GOTO anweisungsnummer,...,
anweisungsnummer...

GOTO hat ein einziges angegebenes Sprungziel, wohingegen ON...GOTO ein Verteilsprung mit mehr als einem Sprungziel sein kann.

Wenn der Verteilsprung ON...GOTO benutzt wird, bestimmt der Wert des ganzzahligen Ausdrucks, zu welcher Anweisungsnummer aus der Liste gesprungen wird.

Beschreibung

Wenn das Sprungziel bei GOTO eine nicht ausführbare Anweisung ist (wie REM), dann wird bei der auf diese Anweisung folgenden Anweisung fortgefahren. Mit GOTO kann nicht in oder aus einer Funktions-Definition (siehe Abschnitt 9) gesprungen werden. Wenn der Sprung auf eine DEF-Anweisung geht, wird auf die erste Zeile hinter der Funktions-Definition gesprungen. (In diesem Fall wird die Funktion redefiniert -- siehe DEF-Anweisung, Abschnitt 9.)

Die Sprungziele in einem Verteilsprung ON...GOTO werden ausgewählt, indem sie, beginnend bei 1, durchnumeriert werden. Es wird dann ausgewählt: die erste Zeilennummer, wenn der Wert des Ausdrucks gleich 1 ist, die zweite, wenn der Wert gleich 2 ist, usw..Wenn der Wert des Ausdrucks kleiner als 1 oder größer als die Zahl der Anweisungsnummer in der Liste ist, wird das ON...GOTO ignoriert und es wird mit der Anweisung direkt hinter ON...GOTO fortgefahren.

Wenn der Ausdruck nicht ganzzahlig ist, wird auf die nächste ganze Zahl gerundet. Dieser Wert wird dann zur Auswahl des Sprungziels benutzt.

Beispiele

Das folgende Beispiel zeigt ein einfaches GOTO in Zeile 45, 55 und 65 sowie einen Verteilsprung in Zeile 30.

```
10 LET I=0
20 LET I=I+1
30 ON I GO TO 40,50,60,70
40 PRINT "DER WERT VON I IST 1"
45 GOTO 20
50 PRINT "DER WERT VON I IST 2"
55 GOTO 20
60 PRINT "DER WERT VON I IST 3"
65 GOTO 20
70 PRINT "DER WERT VON I IST 4"
75 END

*RUN
 DER WERT VON I IST 1
 DER WERT VON I IST 2
 DER WERT VON I IST 3
 DER WERT VON I IST 4
```

Wenn es gestartet wird, druckt das Programm den Wert von I für jedes ON...GOTO.

5.5. GOSUB...RETURN Anweisungen

Mit GOSUB geht die Kontrolle an den Anfang eines einfachen Unterprogramms über. Ein Unterprogramm besteht aus einer Reihe von Anweisungen, die von mehr als einer Stelle im Programm aus ausgeführt werden können. Bei einem einfachen Unterprogramm ist nicht ausdrücklich festgelegt, welche Anweisungen zum Unterprogramm gehören. Eine RETURN Anweisung im Unterprogramm besorgt den Rücksprung zu der Anweisung, die der GOSUB-Anweisung folgt.

Format

GOSUB anweisungsnummer

ON ganzzahliger ausdruck GOSUB anweisungsnummer,...,
anweisungsnummer,...

RETURN

Bei GOSUB ist eine einzige Anweisungsnummer angegeben, während ON...GOSUB wie ein Verteilsprung mehr als eine Anweisungsnummer beinhaltet. Bei Verwendung von ON...GOSUB wird dasjenige Ziel, an das die Kontrolle übergeht, durch den Wert des ganzzahligen Ausdruck bestimmt. Die RETURN-Anweisung besteht einfach aus dem Wort RETURN.

Beschreibung

Ein einfaches GOSUB überträgt die Kontrolle zu der angegebenen Zeile. Bei ON...GOSUB wird das Ziel durch den Wert des ganzzahligen Ausdruck bestimmt. Wie bei ON...GOTO findet kein Unterprogramm-Aufruf statt, wenn der Wert des Ausdrucks kleiner als 1 oder größer als die Länge der Liste ist. Ein GOSUB darf nicht in oder aus einer Funktions-Definition zielen (siehe Abschnitt 9).

Wenn bei der Ausführung innerhalb des Unterprogramms auf eine RETURN-Anweisung gestoßen wird, geht die Kontrolle wieder an die Anweisung hinter dem GOSUB zurück.

Innerhalb eines Unterprogramms kann ein weiteres Unterprogramm aufgerufen werden. Dies wird mit Schachteln bezeichnet. Bei Ausführung eines RETURN wird hinter dasjenige GOSUB zurückgesprungen, das jeweils zuletzt ausgeführt wurde.

Wenn der Ausdruck in einer ON...GOSUB-Anweisung nicht ganzzahlig ist, wird zuerst auf den nächsten ganzzahligen Wert gerundet.

Beispiele

Im ersten Beispiel enthält Zeile 20 eine einfache GOSUB-Anweisung; das Unterprogramm befindet sich in den Zeilen 50 bis 70, mit einem RETURN in Zeile 70.

```
 10 LET B=70
 20 GOSUB 50
 30 PRINT "SINUS VON B IST ";A
 40 GOTO 80
 50 REM:  DIES IST DER ANFANG DES UNTERPROGRAMMS
 60 LET A=SIN(B)
 70 RETURN
 80 REM:PROGRAMM WIRD BEI DER NAECHSTEN ANWEISUNG
       FORTGESETZT
*RUN
 SINUS VON B IST  .7738906815526
```

Die GOSUB-Anweisung kann auf das Unterprogramm folgen, das es aufruft.

```
 10 LET B=70
 20 GOTO 100
 30 REM:  DIES IST DER ANFANG DES UNTERPROGRAMMS
 40 LET A=SIN(B)
 50 RETURN
 60 REM:  HIER KOENNEN WEITERE ANWEISUNGEN STEHEN
 70 REM:  SIE WERDEN NICHT AUSGEFUEHRT
 80 LET A=24, B=50
 90 PRINT "A= ";A,"B= ";B
100 GOSUB 30
110 PRINT "DER SINUS VON B IST ";A
120 REM:  ES SOLLTE GELTEN: A=SIN(B)
130 PRINT "B=";B
140 REM:  ES SOLLTE GELTEN: B=70
*RUN
 SINUS VON B IST  .77389068155526
 B=70
```

Das nächste Beispiel zeigt einen Verteil-Unterprogrammsprung in Zeile 20. Das dritte ausgeführte Unterprogramm enthält einen geschachtelten Aufruf.

```
 10 FOR A=1 TO 3
 20 ON A GOSUB 50,80,110
 30 NEXT A
 40 END
 50 REM:  ERSTES UNTERPROGRAMM
 60 PRINT "ERSTER UP-AUFRUF"
 70 RETURN
 80 REM:  ZWEITES UNTERPROGRAMM
 90 PRINT "ZWEITER UP-AUFRUF"
100 RETURN
110 REM:  DRITTES UNTERPROGRAMM
120 REM:  ES ENTHAELT EINEN GESCHACHTELTEN AUFRUF
130 PRINT "DRITTER UP-AUFRUF"
140 GOSUB 170
145 PRINT "ENDE DES DRITTEN UP-AUFRUFS"
150 RETURN
160 REM:  ANWEISUNG 150 BRINGT RUECKSPRUNG NACH 30
170 REM:  ERSTE ANWEISUNG IN GESCHACHTELTEN AUFRUF
180 PRINT "    GESCHACHTELTER AUFRUF"
190 RETURN
200 REM:  ANWEISUNG 190 BRINGT RUECKSPRUNG NACH 145
*RUN
ERSTER UP-AUFRUF
ZWEITER UP-AUFRUF
DRITTER UP-AUFRUF
    GESCHACHTELTER AUFRUF
ENDE DES DRITTEN  UP-AUFRUFS
```

5.6. Bedingte Anweisung: IF...THEN

Bedingte Anweisungen werden benutzt, um in Abhängigkeit von bestimmten Bedingungen eine Anweisung oder eine Anweisungsfolge abzuarbeiten. Die getestete Bedingung ist ein Zahlwert, der als wahr angesehen wird, wenn er ungleich Null ist, als falsch, wenn er gleich Null ist. Bedingte Anweisungen werden immer durch eine IF-Anweisung eingeleitet. Eine ELSE-Anweisung kann sich anschließen. Sowohl auf THEN- als auch auf ELSE-Anweisungen kann eine Serie von Anweisungen folgen, die in DO und DOEND eingeschlossen sind.

Format

```
IF ausdruck THEN anweisungsnummer
IF ausdruck THEN anweisung
IF ausdruck THEN DO
anweisung
   .
   .
   .
DOEND
```

Einer IF...THEN-Anweisung kann eine ELSE-Anweisung folgen, wenn eine Aktion angegeben werden soll, die durchgeführt wird, falls der Wert der Bedingung falsch ist. Wie bei IF...THEN kann auf ELSE eine Anweisung, eine Anweisungsnummer oder eine Serie von Anweisungen, eingeschlossen in DO...DOEND, folgen.

```
ELSE anweisungsnummer

ELSE anweisung

ELSE DO

anweisung
   .
   .
   .
DOEND
```

ELSE-Anweisungen kommen im Programm nie ohne direkt vorhergehendes IF...THEN bzw. IF...THEN DO...DOEND vor. DO...DOEND-Anweisungen können nur innerhalb von IF...THEN- oder ELSE-Anweisungen benutzt werden.

Die folgenden vier Diagramme zeigen alle möglichen Kombinationen von bedingten Anweisungen. In eckigen Klammern [ ] Eingeschlossenes kann wegfallen. Bei geschweiften Klammern { } muß eine der Möglichkeiten gewählt werden. Anweisungen, die direkt hinter THEN und ELSE stehen, tragen keine Anweisungsnummer. Alle anderen Anweisungen haben eine Nummer.

1) anweisungsnummer IF ausdruck THEN { anweisungsnummer / anweisung }

[anweisungsnummer ELSE { anweisungsnummer / anweisung }]

2) anweisungsnummer IF ausdruck THEN DO

anweisungsnummer anweisung
   .
   .
   .
anweisungsnummer DOEND

[anweisungsnummer ELSE { anweisungsnummer / anweisung }]

3) anweisungsnummer IF ausdruck THEN { anweisungsnummer / anweisung }

```
   anweisungsnummer ELSE DO
   anweisungsnummer anweisung
      .
      .
      .
   anweisungsnummer DOEND
```

4) anweisungsnummer IF ausdruck THEN DO

```
   anweisungsnummer anweisung
      .
      .
      .
   anweisungsnummer DOEND

   anweisungsnummer ELSE DO
   anweisungsnummer anweisung
      .
      .
      .
   anweisungsnummer DOEND
```

Beschreibung

Wenn der Ausdruck hinter IF bei Auswertung wahr ist, dann wird zu der Anweisung gesprungen, deren Nummer hinter THEN steht, bzw. die Anweisung ausgeführt, die hinter THEN steht. Ein Ausdruck wird als wahr angesehen, wenn er Zahlwert hat und dann ungleich Null ist oder wenn er eine Zeichenkette ist und der Wert nicht die Null-Kette ist. Wenn auf THEN ein DO folgt, dann wird die Serie der Anweisungen bis zum DOEND ausgeführt. Danach fährt das Programm normal fort. Wenn der Ausdruck falsch ist, wird mit der nächsten Anweisung fortgefahren bzw. der, die auf das DOEND im Falle von THEN DO folgt.

Wenn auf die IF...THEN-Anweisung eine ELSE-Anweisung folgt, so wird dort die Anweisung oder Anweisungsfolge angegeben, die abgearbeitet werden soll, wenn der Ausdruck falsch ist. Wenn der Ausdruck wahr ist, wird die ELSE-Anweisung bzw. die Gruppe von ELSE-Anweisungen (eingeschlossen durch DO... END) übergangen. Das Programm fährt dann mit der Ausführung der nächsten Anweisung hinter der ELSE-Anweisung bzw. hinter DOEND fort.

Eine FOR-Anweisung kann Bestandteil einer DO...DOEND-Gruppe sein. In diesem Fall muß auch das zugehörige NEXT innerhalb der DO-DOEND-Gruppe liegen (siehe FOR...NEXT-Anweisung, Abschnitt 5.3.).

IF-Anweisungen werden geschachtelt, wenn eine IF-Anweisung in der DO...DOEND-Gruppe einer anderen IF-Anweisung auftritt. In einen solchen Fall gehört jedes ELSE zu dem nächsten vorangegangenen IF, das nicht selbst Teil einer anderen DO...DOEND-Gruppe ist.

Beispiele

Die verschiedenen Typen von IF-Anweisungen werden anhand folgender Beispiele erläutert:

```
10 IF E=F THEN 30
20 LET E=Fx5
30 PRINT E,F
```

Wenn E gleich F ist, geht das Programm zu Zeile 30 über, andernfalls wird in Zeile 20 E auf Fx5 gesetzt und fortgefahren. In beiden Fällen wird Zeile 30 ausgeführt.

```
10 IF X<Y THEN PRINT X
20 ELSE PRINT Y
```

Wenn X kleiner als Y ist, wird der Wert von X ausgegeben, andernfalls wird der Wert von Y ausgegeben. Anschließend fährt das Programm fort.

```
10 IF A<B THEN 100
20 ELSE 200
```

Es wird nach 100 verzweigt, wenn A kleiner als B ist, nach 200, wenn A nicht kleiner als B ist.

```
10 IF K<L THEN LET K=K+1
20 ELSE DO
30   LET L=L/3
40   LET K=LxK
50 DOEND
60 PRINT K,L
```

Wenn K kleiner als L ist, dann wird K um 1 erhöht und nach Zeile 60 verzweigt, wenn K größer oder gleich L ist, wird L auf L/3 und K auf LxK gesetzt und nach Zeile 60 verzweigt.

```
  5 INPUT A
 10 IF A<100 THEN DO
 20   LET A=A+1
 30   GOTO 110
 40 DOEND
 50 ELSE DO
 60   GOSUB 130
 70   LET A=0
 80 DOEND
 90 PRINT "A>=100"
100 END
110 PRINT "A=";A
120 END
130 REM...ANFANG DES UNTERPROGRAMMS
140 PRINT "A=";A
150 RETURN
```

Wenn A kleiner als 100 ist, wird es um 1 erhöht und es wird nach Zeile 110 gesprungen. Wenn A größer oder gleich 100 ist, wird das Unterprogramm bei Zeile 130 ausgeführt. Vom Unterprogramm wird nach Zeile 70 zurückgekehrt, immer noch innerhalb von DO...DOEND.

Wenn ein kleinerer Wert als 100 für A eingegeben wird, wird er um 1 erhöht, Zeile 110 wird ausgeführt und das Programm endet:

```
*RUN
 ?75
 A= 76
```

Wenn ein größerer Wert als 100 für A eingelesen wird, wird das Unterprogramm ausgeführt, anschließend Zeile 100 und das Programm endet:

```
*RUN
?150
A= 150
A>=100
```

Die folgenden Beispiele zeigen geschachtelte IF...THEN Anweisungen.

```
  10 INPUT P,Q,R
  15 PRINT
  20 IF (P+10)=(Q+5) THEN DO
  30   LET P=Q
  40   IF P>R THEN LET P=P-R
  50   ELSE LET P=P+R
  60 DOEND
  70 PRINT P,Q,R
*RUN
?20,25,40
65            25            40
```

```
 10 INPUT A,B,C
 15 PRINT
 20 IF A>B THEN DO
 30   IF B>C THEN DO
 40     IF C=10 THEN DO
 50       LET A=A+1
 60       GOTO 200
 70     DOEND
 80     ELSE GOTO 220
 90   DOEND
100   ELSE DO
110     IF C=10 THEN LET B=C+A
120     ELSE LET C=B-A
130      GOTO 180
140   DOEND
150 DOEND
160 PRINT "A<60>B,A=";A
170 GOTO 230
180 PRINT "A>B,B<60>C,B=";B
190 GOTO 230
200 PRINT "A>B>C,C=10"
210 GOTO 230
220 PRINT "A>B>C,C<60>>10,C=";C
230 END
*RUN
?10,15,20
A<B,A=10

*RUN
?15,5,10
A>B,B<C,B= 25

*RUN
?20,15,5
A>B>C,C<>10,C= 5
```

Zur besseren Übersicht über geschachtelte IF-Anweisungen wurde die Möglichkeit automatischen Einrückens des LIST-Kommandos in den obigen Beispielen zu Hilfe genommen.

5.7. INPUT Anweisung

Die INPUT-Anweisung erlaubt dem Benutzer, Daten von der Tastatur aus einzugeben. Zusätzlich ist es über INPUT möglich, Bestätigungstext auszugeben, der zur Eingabe auffordert.

Format

INPUT liste

INPUT zeichenketten-konstante, liste

Die Elemente der Liste müssen Variablen sein. Die Elemente sind durch Kommas getrennt.

Beschreibung

Bei der Ausführung einer INPUT Anweisung wird über das Dialoggerät ein Fragezeichen (?) ausgegeben und das Programm wartet auf die Eingabe durch den Benutzer. Die Werte sind durch Komma getrennt einzugeben. Wenn eine zu geringe Zahl von Werten eingegeben wird, antwortet das Programm mit einer Meldung, die zur nochmaligen Eingabe auffordert. Der Typ eines Datenelementes, Zahltyp oder Zeichenkette, muß dem Typ der zugeordneten Variablen in der Liste der INPUT Anweisung entsprechen.

Zahlwerte beginnen stets mit den ersten nichtleeren Zeichen vor dem Komma bzw. dem Ende der Eingabezeile.

Zeichenkettenwerte können in Anführungszeichen eingeschlossen sein oder nicht. Wenn nicht, werden Leerzeichen vorn und hinten ignoriert und das Element endet bei einem Komma bzw. dem Ende der Eingabezeile.

Die INPUT Anweisung kann so formuliert werden, daß statt des Fragezeichens ein vorgegebener Text ausgegeben wird. Dieser Text muß dann vor die Liste gesetzt werden. Wenn Eingabe gefordert wird, wird zuerst der Text statt des normalen Fragezeichens ausgegeben.

Beispiele

```
   10  DIM C$[25]
   20  INPUT A,B,C$
   25 PRINT
   30  X=AxB^2
   40  PRINT C$;X
*RUN
?4,7,"X=A MAL B QUADRAT, X="
X=A MAL B QUADRAT, X= 196

   10  INPUT "EINGABEWERT FUER DEN RADIUS ",R
   20  X=3.14xR^2
   30  PRINT "FLAECHE VON X=",X
*RUN
EINGABEWERT FUER DEN RADIUS  25
FLAECHE VON X =      1962.5
```

5.8. PRINT Anweisung und ":"

PRINT bewirkt Daten-Ausgabe über das Dialoggerät. Über den Drucker erfolgt die Ausgabe dann, wenn in der Systemanweisung diese spezifiziert ist (Abschnitt 5.12.). Die auszugebenden Daten werden in einer Liste hinter PRINT aufgeführt.

Format

```
PRINT
:
PRINT liste
: liste
```

Die Liste besteht aus durch Kommas oder Semikolons getrennten Elementen. Die Liste kann durch Komma oder Semikolon abgeschlossen sein. Wird die Liste fortgelassen, bewirkt PRINT einen Zeilenvorschub. Elemente der Liste können Zahlwert- oder Zeichenketten-Ausdrücke sein oder die besondere PRINT-Funktion zur Tabellierung. Das Zeichen ":" kann statt des Schlüsselwortes PRINT verwendet werden.

Beschreibung

Der Inhalt der Liste wird gedruckt. Wenn die Liste mehr als ein Element enthält, müssen Kommas oder Semikolons zur Trennung benutzt werden. Die Wahl zwischen Komma oder Semikolon hängt vom gewünschten Ausgabeformat ab.

Die Ausgabezeile ist in 5 aufeinanderfolgende Felder unterteilt, von denen jedes 14 Zeichen lang ist, so daß sich eine Gesamtlänge von 70 Zeichen ergibt. x) Wenn zwei Elemente durch Komma getrennt sind, wird jedes Element an den Anfang je eines Feldes gedruckt. Bei Trennung durch Semikolon werden die Elemente unmittelbar aufeinander folgend gedruckt. In beiden Fällen wird das Element auf die folgende Zeile gebracht, wenn in der alten Zeile nicht mehr genügend Platz für das ganze Element ist.

x) ANMERKUNG: Die Voreinstellung für die Ausgabe-Zeilenlänge kann durch einen Aufruf durch SYSTEM "LINELEN" geändert werden (siehe Abschnitt 5.12.).

Wagenrücklauf (CR) und Zeilenvorschub (LF) werden nach Ausführung von PRINT ausgeführt, es sei denn, die Liste ist durch Komma oder Semikolon abgeschlossen. In diesem Falle beginnt die nächste PRINT Anweisung auf derselben Zeile.

Wenn in der Liste ein Ausdruck auftritt, wird er zuerst ausgewertet und das Ergebnis wird gedruckt. Jeder Variablen muß vor der Ausgabe ein Wert zugewiesen worden sein. Jedes Zeichen zwischen Anführungszeichen in einer Zeichenketten-Konstanten wird gedruckt.

Zahlenwerte werden linksbündig in ein Feld gelegt, dessen Breite sich aus der Größe der Zahl ergibt. Die Breite beinhaltet eine Stelle links für das mögliche Vorzeichen. (So wird ein Leerzeichen gedruckt, falls die Zahl nicht-negativ ist.)

Beispiele

Im folgenden Beispiel bewirkt die erste PRINT Anweisung die Auswertung und Ausgabe von drei Ausdrücken. Das zweite PRINT bewirkt Zeilenvorschub. Die dritte und vierte PRINT Anweisung kombinieren eine Zeichenketten-Konstante mit einem Zahlwert-Ausdruck. In der Druckzeile werden keine Felder benutzt, wenn nicht das Komma als Trennzeichen verwendet wird. Die vierte PRINT Anweisung druckt in dieselbe Zeile wie zuvor die dritte, weil die dritte PRINT Anweisung mit einem Komma abschließt.

```
  10  LET A=1,B=2,C=3,D=4,E=5
  20  PRINT A,CxD,E-BxB
  30  PRINT
  40  PRINT "A/(B-C)=";A/(B-C),
  50  PRINT "E+D=";E+D
*RUN
 1            12          1

A/(B-C)=-1    E+D= 9
```

5.8.1. TAB Funktion

TAB (n)

TAB bringt den Cursor auf Spalte n MOD (70). Wenn die augenblickliche Cursor-Position größer als n MOD (70) ist, geht der Cursor auf die Position n MOD (70) auf der nächsten Zeile.

Bemerkung: Wenn die voreingestellte Zeilenlänge mit SYSTEM "LINELEN" abgeändert wurde (siehe Abschnitt 5.12.), bezieht sich alles oben gesagte auf (Zeilenlänge) statt auf (70).

Beispiele

```
  10 PRINT "123456789";TAB(4);"ABCD"
*RUN
123456789
   ABCD
```

Die Cursor-Position ist größer als vier. Deshalb geht der Cursor in die Position vier der nächsten Zeile. Die Zeichenkette "ABCD" beginnt deshalb in der fünften Position.

5.9. READ/DATA/RESTORE Anweisungen

Die READ, DATA und RESTORE Anweisungen stellen zusammen eine Methode der Dateneingabe an ein BASIC Programm dar. Die READ Anweisung liest Daten, die in DATA Anweisungen angegeben sind, in Variablen, die in der READ Anweisung aufgeführt sind. RESTORE ermöglicht es, dieselben Daten noch einmal zu lesen.

Format

READ list

Die Elemente in der Liste sind Variablen. Elemente sind durch Komma voneinander getrennt.

DATA konstante,...,konstante

Die Konstanten haben entweder Zahlwert oder sind Zeichenketten. Konstanten in der DATA Anweisung werden den Variablen in der READ Anweisung zugewiesen. Die Zuordnung geschieht in der Reihenfolge der Anordnung beider Listen: Die erste Konstante zur ersten Variablen, die zweite Konstante zur zweiten Variablen, und so weiter. Ein Kommentar (Abschnitt 5.10.) kann in einer DATA Anweisungszeile NICHT angegeben werden.

RESTORE

RESTORE zeilennummer

ON ganzzahliger ausdruck RESTORE zeilennummer,...,
zeilennummer

Die Zeilennummern bezeichnen DATA Anweisungen.

Beschreibung

Bei Ausführung einer READ Anweisung wird jeder Variablen

ein neuer Wert aus der Konstanten-Liste einer DATA Anweisung zugewiesen. RESTORE ermöglicht es, die erste Konstante erneut zuzuweisen, wenn die nächste READ Anweisung ausgeführt wird, bzw. wenn eine Zeilennummer bei RESTORE angegeben ist, die erste Konstante einer bestimmten Konstantenliste.

Es kann mehr als eine DATA Anweisung angegeben werden. Alle Konstanten in diesen DATA Anweisungen bilden dann eine kombinierte Datenliste. Die Liste fängt mit der DATA Anweisung mit der kleinsten Anweisungsnummer an und setzt sich fort bis zur Anweisung mit der größten Nummer. DATA Anweisungen können sich überall im Programm befinden. Sie müssen weder vor den READ Anweisungen stehen, noch müssen sie direkt aufeinander folgen.

Wenn eine Variable Zahlwert hat, muß das entsprechende Element der Datenliste ebenfalls Zahlwert haben. Wenn eine Variable eine Zeichenkette ist, kann das entsprechende Element beliebig sein.

Für die Datenliste gibt es einen Zeiger, der festhält, welche Konstante als nächste zur Zuweisung an eine Variable ansteht. Dieser Zeiger beginnt mit der ersten DATA Anweisung und wird Schritt für Schritt durch die Datenliste gezogen. Mittels der RESTORE Anweisung ist ein nicht serieller Zugriff möglich. Die übliche Reihenfolge wird durchbrochen, indem der Zeiger auf eine ausgewählte DATA Anweisung gesetzt wird.

Wenn in der RESTORE Anweisung eine Zeilennummer angegeben ist, wird der Zeiger auf die erste Konstante in der angegebenen DATA Anweisung gesetzt. Wenn keine Zeilennummer angegeben ist, wird der Zeiger auf die erste Konstante in der ersten DATA Anweisung des Programms gesetzt.

Eine von vielen Zeilennummern kann mittels ON...RESTORE Anweisung ausgewählt werden. Der Ausdruck wird auf die nächste ganze Zahl gerundet und eine Zeilennummer wird aus der Liste in der unter ON...GOTO (Abschnitt 5.4.) bereits beschriebene Weise ausgewählt. Der Zeiger wird auf die erste DATA Anweisung hinter der ausgewählten Zeilennummer gesetzt. Wenn der Ausdruck nicht zur Auswahl benutzt werden kann, wird der Zeiger nicht versetzt.

Die Daten in Anweisung 10 werden in Anweisung 20 gelesen und in Anweisung 30 gedruckt:

```
  10  DATA 3,5,7
  20  READ A,B,C
  30  PRINT A,B,C
*RUN
 3             5             7
```

Im folgenden Beispiel werden dieselben Konstanten nochmal in einen zweiten Satz von Variablen gelesen:

```
  5  DIM A$[3],C$[3],D$[3],E$[3],B$[3]
 10  DATA 3,5,7
 20  READ A,B,C
 30  READ A$,B$
 40  DATA ABC,DEF
 50  RESTORE 30
 60  READ C$,D$
 70  RESTORE
 80  READ D,E,F,E$
 90  PRINT A,B,C
100  PRINT A$+B$,C$+D$
110  PRINT D,E,F,E$

*RUN
 3          5          7
ABCDEF     ABCDEF
 3          5          7          ABC
```

Im folgenden Beispiel wird ein ON...RESTORE zusammen mit einem ON...GOTO verwendet.

```
  10 DIM A%(10),B%(10),C%(10),D%(10)
  20 DATA 1111
  30 DATA 2222
  40 DATA 3333
  50 DATA 4444
  60 LET I=0
  70 IF I=4 THEN END
  90 PRINT
 100 ON I RESTORE 30,40,50
 110 ON I GOTO 130,140,150
 120 READ A%
 125 PRINT A%;
 130 READ B%
 135 PRINT B%;
 140 READ C%
 145 PRINT C%;
 150 READ D%
 155 PRINT D%;
 160 LET I=I+1
 165 GOTO 70
*RUN
 1111222233334444
 222233334444
 33334444
 4444
```

5.10 Kommentare: REM Anweisung und "'"

Die REM Anweisung erlaubt die Einfügung einer Kommentarzeile in die Auflistung des Programms. Die Kommentare beeinflussen den Programmlauf nicht.

Format

REM beliebige Zeichen

Wie bei anderen Anweisungen steht vor REM eine Anweisungsnummer

Beschreibung

Die Kommentare innerhalb von REM Anweisungen werden als Teil des BASIC Programms gerettet und gedruckt, wenn das Programm gelistet wird. Sie werden ignoriert bei Ausführung des Programms.

Kommentare sind leichter zu lesen, wenn auf das REM Leerzeichen folgen oder ein Doppelpunkt wie in den Beispielen gezeigt.

Kommentare können auch hinter jede Anweisung gesetzt werden, indem zwischen Anweisung und Kommentar ein "\" gesetzt wird. Zu beachten ist, daß bei Eingabe des Kommentars nach einer Anweisung zwei "\" Zeichen eingetastet werden.

Kein Kommentar darf hinter eine DATA Anweisung gesetzt werden (Abschnitt 5.9.).

Beispiele

```
10 REM: DIES IST EIN BEISPIEL
20 REM  FUER REM ANWEISUNGEN
30 REM -- HIER KOENNEN BELIEBIGE ZEICHEN STEHEN "/x!&,
40 REM...REM ANWEISUNGEN WERDEN NICHT AUSGEFUEHRT
50 PRINT A+B \ HIER FOLGT EIN KOMMENTAR AUF EIN PRINT
```

5.11. RANDOMIZE Anweisung

Die RANDOMIZE Anweisung wird benutzt, um die Ausgangszahl zu ändern, die von der Funktion RND (Abschnitt 6.8.7.) zur Erzeugung von Pseudozufallszahlen verwendet wird.

Format

RANDOMIZE

Beschreibung

Es gibt 128 verschiedene Ausgangszahlen. Die RANDOMIZE Anweisung kann überall ins Programm gesetzt werden. Bei jeder Ausführung wird die Reihe der Zufallszahlen, die durch RND erzeugt wird, mit einer neuen Ausgangszahl begonnen. Durch RANDOMIZE werden die Zufallszahlen erst richtig "zufällig". Sonst würde bei jedem Programmlauf stets die gleiche Reihe Zahlen durchlaufen.

5.12 SYSTEM Anweisung

Die SYSTEM Anweisung wird zur Durchführung verschiedener Funktionen benutzt, die nicht in ein maschinen-unabhängiges BASIC gehören.

Format

SYSTEM operation[,parameterliste]

Operation ist ein Zeichenkettenausdruck, dessen Wert eine der unten aufgeführten Systemoperationen ist.

Beschreibung

Die verfügbaren Operationen bestimmen die Ausgabe von Warnungen, das Einrücken bei Auflistung, das Einrücken bei ASAVE Dateien, Druckzeilenlänge und die Ausgabe über Drucker. Die Voreinstellungen sind folgende:

Warnungsausgabe	:	ON (EIN)
Einrücken bei LIST	:	ON (EIN)
Einrücken bei ASAVE	:	OFF (AUS)
Druckzeilenlänge	:	70
Ausgabe über Drucker	:	OFF (AUS)

Die erlaubten Operationen und deren benötigte Parameter sind folgende:

Operation: "WARNOFF"
Parameter: Keine
Funktion: Unterdrückt die Ausgabe aller Warnungen, die während einer BASIC Sitzung erzeugt werden.

Beispiel: SYSTEM "WARNOFF"

Operation: "WARNON"
Parameter: Keine
Funktion: Ermöglicht die Ausgabe von Warnungen.

Beispiel: SYSTEM "WARNON"

Operation: "INDENTOFF"
Parameter: Keine
Funktion: Stellt das normalerweise durchgeführte Einrücken bei der Ausführung des LIST Kommandos ab.

Beispiel: SYSTEM "INDENTOFF"

Operation: "INDENTON"
Parameter: Keine
Funktion: Stellt das Einrücken bei LIST an.

Beispiel: SYSTEM "INDENTON"

Operation: "ASINOFF"
Parameter: keine
Funktion: Stellt das Einrücken der Zeilen von ASAVE Dateien ab.

Beispiel: SYSTEM "ASINOFF"

Operation: "ASINON"
Parameter: keine
Funktion: Stellt das Einrücken der Zeilen von ASAVE Dateien an.

Beispiel: SYSTEM "ASINON"

Operation: "LINELEN"
Parameter: Zeilenlänge (ganze Zahl zwischen 1 und 255)
Funktion: Setzt die Zeilenlänge. Ausgaben, die nicht mehr zwischen die augenblickliche Cursor-Position und das Zeilenende passen, kommen in die nächste Zeile. Voreinstellung ist 70.

Beispiel: SYSTEM "LINELEN", 132
Die Druckzeilenlänge für 132 Zeichen breites Papier wird angegeben.

Operation: "LPRINTOFF"
Parameter: keine
Funktion: Dialogausgabe erfolgt nur über Bildschirm

Beispiel: SYSTEM "LPRINTOFF"

Operation:	"LPRINTON"
Parameter:	keine .
Funktion:	Dialogausgabe über Drucker und zusätzlich über Bildschirm
Beispiel:	SYSTEM "LPRINTON"

6. Zahlenwerte und Wertarten

SIOS-BASIC hat für die Darstellung der Zahlenwerte die Wertarten Gleitkomma (reelle) Zahl und ganze Zahl. Von diesen Typen existieren in der Sprache Variable, Felder, Konstanten, Ausdrücke, Zuweisungen und Funktionen.

6.1. Typenspezifikation

Zahlenwert-Variablen und -Felder haben jeweils ihren eigenen Datentyp. Ein an den Variablennamen angehängtes Zeichen bestimmt diesen Typ. Variablen mit gleichem Namen aber verschiedenem Anhang sind voneinander unterschieden.

Beschreibung

Variablen ohne angehängtes Zeichen sind vom Typ REAL (reelle Zahl). In Abschnitt 6.2.2. findet sich eine Beschreibung der Darstellung von Gleitkommazahlen.
Variablen mit dem Anhangszeichen "%" sind vom Typ INTEGER (ganze Zahl). Der Wertebereich der ganzen Zahlen geht von - 32768 bis 32767 (2^{16} verschiedene Werte).
Variablen mit dem Anhangzeichen "$" haben Zeichenkettenwert (siehe Abschnitt 8).

6.2. Form von Zahlenwertkonstanten

Wenn Konstanten in Ausdrücken, DATA Anweisungen oder bei Ausführung von INPUT Anweisungen auftreten, sind sie in einer von drei Formen dargestellt: INTEGER, Festkomma oder Gleitkomma. Fest- und Gleitkomma-Zahlen sind vom Typ REAL.

6.2.1. Ganze Zahlen

Eine ganze Zahl ist eine Folge von Ziffern ohne einen Dezimalpunkt. Beispiele für diese Formen von Konstanten:

```
10  LET A%=47 ,B%=-375,C%=607 ,D%=0
20  PRINT A%,B%,C%,D%
* RUN
47          -375          607          0
```

Eine vorzeichenlose, ganzzahlige Konstante, die kleiner als 256 ist, wird intern als INTEGER dargestellt. Alle anderen Zahlwertkonstanten werden als REAL dargestellt.

Wenn Rechenoperationen auf Ausdrücken ausgeführt werden, die nur INTEGER-Konstanten und -Variablen enthalten, sind die Ergebnisse auch ganzzahlig (von Typ INTEGER).
Wenn jedoch ein beteiligter Operand vom Typ REAL ist, ist das Ergebnis REAL.

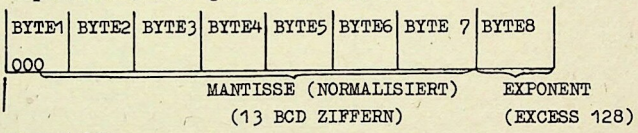
6.2.2. Gleitkommazahlen

Eine Gleitkommazahl ist eine Zahl, die intern als ein Bruch (genannt Mantisse) und eine Hochzahl (genannt Exponent) abgespeichert ist.

.3E-11 .3x10^(-11)

Dezimale Gleitkommadarstellung

13 BCD-Ziffern normalisierte[x] Mantisse mit Vorzeichen
8 Bit excess-128 Exponent (zur Bais 10)
Das Vorzeichen-Bit ist die höherwertigste Ziffer
Exponent=0 heißt: ganzer Wert ist 0.



Vorzeichen
0=+
1=-

```
09 99 99 99 99 99 99 FF = .99999 99999 999 x 10^127
01 00 00 00 00 00 00 81 = 1.0
01 00 00 00 00 00 00 01 = .1   10 (427)
XX XX XX XX XX XX XX 00 = 0
89 99 99 99 99 99 99 FF = -.99999 99999 99 x 10^127
```

alle Ergebnisse werden auf Mantissen von 13 BCD-Ziffern gekürzt.
Das folgende Beispiel weist zwei reellen Variablen Werte zu und druckt diese dann aus:

```
10  LET I=2795448.6 , J=2.79E-3
20  PRINT I,J
*RUN
2795448.6       .00279
```

x Normalisiert heißt, daß der Exponent so eingestellt wird, daß entweder das höherwertigste Zeichen der Mantisse ungleich 0 ist, oder die ganze Mantisse leer ist.

xx Excess 128 heißt, daß die Hochzahl für die jeweilige Basis gleich ist dem Wert des Exponentenfeldes minus 128.

6.3. Zahlenwert-Ausdrücke

Variablen jeden Datentyps und Zahlen jeder Datenform können in Zahlenwert-Ausdrücken benutzt werden. SIOS-BASIC stellt Rechenoperationen für beide Datentypen zur Verfügung, sowie automatische Typumwandlung, wenn zwei Operanden nicht vom gleichen Typ sind. Die folgende Tafel faßt die Typen der Ergebnisse zusammen, die bei Verknüpfung von Rechengrößen durch beliebige Operatoren (außer &, !, –, /, ^, und Vergleichen) entstehen.

TAFEL 6-1

Type der ersten Rechengröße	Typ der zweiten Rechengröße: INTEGER	Typ der zweiten Rechengröße: REAL
INTEGER	INTEGER	REAL
REAL	REAL	REAL

Wenn die Operatoren &, !, ¯, =, <,> , <=,> = und < > benutzt werden, ist das Ergebnis immer vom Typ REAL (0 für falsch, 1 für wahr). Wenn die Operatoren / und ^ verwendet werden, ist das Ergebnis immer vom Typ REAL.

Beispiel

Eine ganze Zahl, die mit einer reellen Zahl in einem Ausdruck verknüft wird, ergibt eine reelle Zahl; zwei ganze Zahlen ergeben wieder eine ganze Zahl:

```
 10  LET  I%=25,I1%=50,R=2.75
 20  PRINT I%+I1%
 30  PRINT I%+R
*RUN
 75
 27.75
```

6.4. Zahlenwert-Ausdrücke als Bedingungen

Die Zahlwertausdrücke, die bei Sprungentscheidungen in bedingten Ausdrücken (siehe Abschnitt 5.6.) auftreten, können jeden beliebigen Datentyp enthalten. Das Ergebnis wird in den Typ REAL umgewandelt. Der Ausdruck wird dann als falsch angenommen, wenn er gleich 0 ist, sonst als wahr.

6.5. Zahlenwertzuweisung

Wenn das Ergebnis eines Zahlenwert-Ausdrucks einer Variablen zugewiesen wird, wird es in den Typ dieser Variablen umgewandelt.

Die Methode der Umwandlung bei Zuweisung von Werten an Variablen unterschiedlichen Typs ist in Tafel 6-2 zusammengefaßt:

TAFEL 6-2

Variablentyp	Wertetyp	Umwandlungsmethode
INTEGER	REAL	RUNDEN
REAL	INTEGER	GLEITEN

Diese Tafel findet überall Anwendung, wo Werte an Variablen zugewiesen werden (INPUT, READ, etc.).

6.6. Eingabe von Zahlenwerten

Konstanten jeder Datenform können mit Hilfe von READ und INPUT Anweisungen eingegeben werden. Einmal eingelesen, werden sie nach Tafel 6-2 an den Typ der aufnehmenden Variablen angeglichen.

6.7. Ausgabe von Zahlenwerten

Zahlen jeden Datentyps können mittels der PRINT USING Anweisung ausgegeben werden (siehe Abschnitt 11). Zahlen jeden Datentyps können auch in Dateien auf Massenspeichern geschrieben werden. Dieser Prozeß ist ausführlich in Abschnitt 10 beschrieben.

6.8. Zahlenwertfunktionen

Die meisten eingebauten Funktionen, die einen Zahlwert zurückgeben, geben einen vom Typ REAL zurück. (Benutzerdefinierte Zahlwertfunktionen sind in Abschnitt 9 beschrieben). Diese Werte werden wie in Tafel 6-2 beschrieben umgewandelt, wenn sie in Ausdrücken oder Zuweisungen verwendet werden.
Zahlwert-Argumente für Funktionen können sowohl vom Typ REAL als auch vom Typ INTEGER sein und werden jeweils dem

für den Parameter geforderten Typ angepaßt. Dies geschieht nach Tafel 6-2.

6.8.1. ABS Funktion

ABS(ausdruck)

ABS liefert den Absolutwert des Ausdrucks.

6.8.2. ATN Funktion

ATN(ausdruck)

ATN liefert den Arcustangens des Ausdrucks. Das Ergebnis ist ein Winkel-Bogenmaß im Bereich von -pi/2 bis pi/2.

6.8.3. COS Funktion

COS(winkel in bogenmaß)

COS liefert den Kosinus des Winkels MOD 2pi.

6.8.4. EXP Funktion

EXP(ausdruck)

EXP liefert e^ausdruck, wobei e die Eulersche Zahl ist, 2.718281828.

6.8.5. INT Funktion

INT(ausdruck)

INT liefert die größte ganze Zahl, die kleiner oder gleich dem Ausdruck ist.

6.8.6. LOG Funktion

LOG(ausdruck)

LOG liefert den natürlichen Logarithmus des Ausdrucks. Der Ausdruck muß größer als 0 sein.

6.8.7. RND Funktion

RND

RND liefert eine Pseudo-Zufallszahl größer oder gleich 0 und kleiner als 1. Die Ausgangszahl für RND kann durch die RANDOMIZE Anweisung (Abschnitt 5.11.) geändert werden.

6.8.8. SGN Funktion

SGN(ausdruck)

SGN liefert das Vorzeichen des Ausdrucks. Wenn ausdruck > 0, ergibt sich 1. Wenn ausdruck=0, ergibt sich 0. Wenn ausdruck < 0, ergibt sich -1.

6.8.9. SIN Funktion

SIN(winkel in bogenmaß)

SIN liefert den Sinus des Winkels MOD 2pi.

6.8.10. SQR Funktion

SQR(ausdruck)

SQR liefert die Quadratwurzel des Ausdrucks. Der Ausdruck muß größer oder gleich 0 sein.

6.8.11. TAN Funktion

TAN(winkel in bogenmaß)

TAN liefert den Tangens des Winkels MOD 2pi.

7. Felder

Ein Feld ist eine Menge von Variablen, die unter einem gemeinsamen Namen abgespeichert ist. Die einzelnen Elemente eines Feldes werden durch Zusatz einer Indizierung an den Namen ausgewählt: z.B.: A [7] ist das siebte Element des Feldes A.

Felder dürfen in BASIC ein oder zwei Dimensionen haben. Bei einem eindimensionalen Feld werden die Elemente durch einen einzelnen Index ausgewählt, der die Zeilennummer angibt. Zeilen und Spalten sind ab 1 durchnumeriert. Ein zweidimensionales Feld besteht aus einer angegebenen Zahl von Zeilen und einer angegebenen Zahl von Spalten.
Z.B. kann ein Feld mit vier Zeilen und drei Spalten wie folgt dargestellt werden:

Spalten

Zeilen	1	2	3
1	A[1,1]	A[1,2]	A[1,3]
2	A[2,1]	A[2,2]	A[2,3]
3	A[3,1]	A[3,2]	A[3,3]
4	A[4,1]	A[4,2]	A[4,3]

Jedes Element des Feldes wird durch ein Indexpaar, getrennt durch Komma, bestimmt. Der erste Index zeigt die Zeile, der zweite die Spalte an.
Jedes Feld in einem SIOS BASIC Programm ist auf eine von zwei Weisen definiert:

Durch eine DIM Anweisung, die den Namen des Feldes sowie die Zahl der Zeilen und Spalten bestimmt.

Durch Gebrauch: Zahlwertfelder, die benutzt, aber nicht in einer DIM Anweisung ausdrücklich definiert sind, haben 10 Zeilen, wenn eindimensional, oder 10 Zeilen und 10 Spalten, wenn zweidimensional.

Die physische Größe eines Feldes ist die Gesamtzahl der vom Feld ursprünglich belegten Elemente; die logische Größe ist die augenblickliche Zahl der Zeilen, multipliziert mit der augenblicklichen Zahl der Spalten.
Die physische Größe eines Feldes kann während der Ausführung nicht geändert werden.

Anders die logische Größe (d.h.: Zahl der Zeilen und Spalten): sie kann durch eine DIM Anweisung geändert werden, solange die physische Größe nicht überschritten wird.
SIOS BASIC erlaubt Felder für jeden Zahlwert-Datentyp sowie eindimensionale Zeichenketten-Felder. Bemerkungen in diesem Kapitel beziehen sich, wenn nicht anders vermerkt auf Zahlwert-Felder. Zeichenketten-Felder werden in Abschnitt 8 beschrieben.
Dieser Abschnitt beschreibt DIM Anweisungen, wie sie für Zahlwert-Felder benutzt werden.
Runde und eckige Klammern sind gleichwertig im Zusammenhang mit der Angabe von indizierten Feldvariablen verwendbar.

7.1. DIM Anweisung

Die DIM Anweisung wird benutzt, um Speicher für Felder zu reservieren und um obere Grenzen für die Zahl der Elemente von Feldern zu setzen. DIM Anweisungen können ebenso für Zeichenketten verwendet werden (siehe Abschnitt 8). Dieser Abschnitt bezieht sich nur auf Zahlwert-Felder.

Format

DIM variable [ganze zahl],...,variable [ganze zahl]

wobei variable der Feldname ist und die ganze Zahl die Zahl der Zeilen eines eindimensionalen Feldes angibt.

DIM variable [ganze zahl,ganze zahl],...,
variable [ganze zahl,ganze zahl]

wobei variable der Name eines zweidimensionalen Feldes ist, die erste ganze Zahl die Zahl der Zeilen und die zweite die Zahl der Spalten bestimmt.
Zeilen und Spalten werden von 1 an durchnumeriert. Die Gesamtgröße ist die Zahl der Elemente. In einem eindimensionalen Feld ist sie identisch mit der Zeilenzahl: in einem zweidimensionalen Feld ist sie das Produkt von Zeilen- und Spaltenzahl.
Sowohl ein- als auch zweidimensionale Zahlwert-Felder und

Zeichenkettenvariablen können in derselben DIM Anweisung deklariert werden, sie werden durch Komma getrennt. Jedes Element eines Zahlwert-Feldes wird auf 0 gesetzt.

Beschreibung

Die Elemente eines Feldes werden durch indizierte Variablen angesprochen. Die Werte der Elemente sind 0 nach Ausführung der DIM Anweisung. Die Zahl der Elemente des Feldes wird durch eine DIM Anweisung oder durch Gebrauch definiert. Die DIM Anweisung kann überall im Programm auftreten und ausgeführt werden. Auf diese Weise muß die DIM Anweisung ausgeführt werden, bevor auf das Feld zugegriffen wird.

Beispiel

```
10 DIM A[17],A7[6,8],B[2,5]
20 REM   A HAT 17 ZEILEN, EINE SPALTE
30 REM   A7 UND B SIND ZWEIDIMENSIONALE FELDER
40 REM   A7 HAT 6 ZEILEN, 8 SPALTEN;B HAT 2 ZEILEN, 5 SPALT.
50 DIM C[5],C1[5,1],C2[1,5]
60 REM   C UND C1 HABEN DIESELBEN DIMENSIONEN:5 ZEIL.,1SPAL.
70 REM   C2 HAT 1 ZEILE, 5 SPALTEN
```

Wichtig ist, daß die DIM Anweisung für C1 in Zeile 50 etwas anderes bewirken würde, wenn hier C1[5] stehen würde, da auf Feldelemente mit derselben Zahl von Indizes zugegriffen werden muß, wie in der DIM Anweisung angegeben.
Die DIM Anweisung kann benutzt werden, um die Zahl der Zeilen und Spalten eines schon existierenden Feldes zu ändern. Dabei muß folgendes beachtet werden:

1) Die Zahl der Dimensionen muß die gleiche bleiben.

2) Die Gesamtzahl der Elemente (Zeilen mal Spalten) darf nicht über die physische Größe (ursprüngliche Zuweisung) hinausgehen.

3) Der Typ muß "Zahlwert" sein. Zeichenketten-Felder können nicht redimensioniert werden.

Wichtig: Alle Daten im Feld gehen verloren, da das Feld bei Redimensionierung wieder mit 0 besetzt wird.

7.2. Speicherung von Daten in Feldern

Es gibt mehrere Methoden, Feldern Werte zuzuweisen. Einzelne Elemente können in einer Zuweisungsanweisung angesprochen werden:

```
10  LET A[5]=26
20 A7[1,6]=Nx4.5
```

Zusätzlich können einzelne Elemente in INPUT und READ Anweisungen auftreten:

```
10 INPUT A[1],A[2],A[3]
20 READ A7[3,2]
```

FOR Schleifen können zur elementweisen Besetzung ganzer Felder verwendet werden:

```
10 DIM A[17], B[6,8]
20 FOR N=1 TO 17
30   INPUT A[N]
40 NEXT N
50 FOR N=1 TO 6
60   FOR M=1 TO 8
70     READ B[N,M]
80   NEXT M
90 NEXT N
```

7.3. Ausgabe von Daten aus Feldern

Die Mechanismen zur Ausgabe von Daten aus Feldern sind entsprechend denen zur Datenzuweisung an Feldern. Einzelne Elemente können mit PRINT ausgegeben werden:

```
100 PRINT A[1],A[2],A[3]
```

FOR Schleifen können zur elementweisen Ausgabe ganzer Zahlen verwendet werden:

```
90 DIM A[17], B[6,8]
```

```
100 FOR N=1 TO 17
110   PRINT A[N]
120 NEXT N
130 FOR N=1 TO 6
140   FOR M=1 TO 8
150     PRINT B[N,M]
160   NEXT M
170 NEXT N
```

8. Zeichenketten

SIOS BASIC erlaubt dem Programmierer die Bearbeitung von Zeichenketten durch Verwendung von Zeichenketten-Konstanten, Variablen, Feldern, Funktionen, Operatoren, Zuweisungs-Anweisungen und Ein/Ausgabe Anweisungen.
Viele dieser Einsatzgebiete von Zeichenketten sind Erweiterungen von Anweisungen, die schon beschrieben wurden (wie z.B. READ und PRINT).

8.1. Zeichenketten-Konstanten

Eine Zeichenketten-Konstante ist eine Folge von bis zu 255 Zeichen. Jedes Zeichen wird intern durch eine Zahl zwischen 0 und 255 dargestellt, wie das durch den KOI-7-Bit-Code festgelegt ist (siehe AnhangA).
Einige dieser Zeichen sind graphisch darstellbar (man kann sie drucken - A,B,d,%, wohingegen andere es nicht sind (man kann sie nicht drucken - RETURN, LF). Beide Typen von Zeichen können in Zeichenketten-Konstanten verwendet werden, sie werden jedoch unterschiedlich behandelt.

Format
Eine Zeichenketten-Konstante besteht aus einer Folge von darstellbaren Zeichen, eingeschlossen in Anführungszeichen:

"darstellbare zeichen"

Das Anführungszeichen (") und die linke spitze Klammer (<) können innerhalb einer solchen Folge nicht in ihrer eigentlichen (druckbaren) Darstellung auftreten.
Das Anführungszeichen, die linke spitze Klammer und nicht-darstellbare Zeichen können jedoch auch in einer Zeichenketten-Konstanten untergebracht werden, indem die entsprechende Dezimalzahl für das KOI-7-Bit-Codezeichen in spitze Klammern eingeschlossen wird.

<ganze zahl>

Die ganze Zahl ist der Dezimalwert des gewünschten Zeichens und kann aus dem Bereich 0 bis 255 sein, Man sollte jedoch die Benutzung dieser Form auf nicht-darstellbare Zeichen, das Anführungszeichen (34) und die linke spitze Klammer beschränken. Nicht-darstellbare Zeichen können mit den anderen gemeinsam in einer Zeichenkette verwendet werden.

Beschreibung

Zeichenketten-Konstanten können sowohl Groß- als auch Kleinbuchstaben beinhalten. Wenn eine Zeichenketten-Konstante gedruckt wird, wird jeder Zeichenwert seiner Bedeutung entsprechend auf dem Ausgabegerät gedruckt. Andererseits wird bei Auflistung eines Programms jedes darstellbare Zeichen einer Zeichenketten-Konstante (ausgenommen " und<) in der graphischen Darstellung und jedes nicht-darstellbare Zeichen in der Spitze-Klammer-Form gedruckt. Zeichen, die in einer Zeichenkette-Konstanten graphisch dargestellt wurden, sind intern so dargestellt, daß ihr höher-wertigstes Bit auf 0 gesetzt ist; d.h. sie werden durch KOI-7-Bit-Code-Werte kleiner als 128 dargestellt.

Beispiele

""	Eine Null-Kette (eine Zeichenkette der Länge 0)
"BASIC"	
"B "	
"<13><10>"	Wagenrücklauf, Zeilenvorschub
"ZEILE 1<13><10>ZEILE 2"	Zweizeiliger Text
"A<124>B"	A, senkrechter Strich, B
"<34>"	ein Anführungszeichen

8.2. Zeichenketten-Variablen

Eine Zeichenketten-Variable (einfach oder indiziert) wird zur Aufnahme von Folgen von KOI-7-Bit-Code-Zeichen benutzt. Die vereinbarte Größe der Zeichenketten-Variable wird deren physische Länge genannt. Die maximale Länge jeder Zeichenketten-Variablen ist 32767 Zeichen. Zeichenketten-Variablen sind ferner dadurch beschränkt, daß sie in den verfügbaren Hauptspeicher passen müssen.
Während der Ausführung enthält jede Zeichenketten-Variable Zeichenketten, deren Länge die physische Länge der Variablen nicht überschreiten kann. Diese dynamische Länge wird die logische Länge der Variablen genannt und mit 0 initialisiert. Jede Zeichenketten-Variable enthält also bei Beginn der Programmausführung die leere Kette.
Einfache und indizierte Zeichenketten-Variablen müssen durch eine DIM Anweisung (Abschnitt 8.3.) dimensioniert werden.

Format

Eine einfache Zeichenketten-Variable wird durch ihren Namen referiert, bei Bedarf durch eine Teilketten-Beschreibung, die in Klammern oder eckige Klammern eingeschlossen ist.

zeichenketten-name
zeichenketten-name [erstes zeichen]
zeichenketten-name [erstes zeichen,letztes zeichen]

zeichenketten-name ist ein Buchstabe gefolgt von "$". oder ein Buchstabe, eine Ziffer und ein "$".
Die Teilketten-Beschreibung besteht aus einem oder zwei Zahlwert-Ausdrücken, die durch Komma getrennt sind. Der erste Ausdruck gibt stets die Position des letzten Zeichens an. Wenn nur ein Ausdruck angegeben ist, geht die Teilkette bis zum Ende der ganzen Zeichenkette.
Ein Zeichenketten-Feld wird durch den Zeichenketten-Namen referiert, gefolgt von dem Index und bei Bedarf einer durc. Komma abgetrennten Teilketten-Beschreibung. Die beiden letzten Angaben werden von Klammern oder eckigen Klammern umschlossen.

zeichenketten-name [index]
zeichenketten-name [index,erstes zeichen]
zeichenketten-name [index,erstes zeichen,letztes zeichen]

Der Index ist ein ganzzahliger Ausdruck, der das Element des Feldes auswählt. Weil ein Zeichenketten-Feld nur eine Dimension haben darf, tritt höchstens ein Index auf.
Die Teilketten-Beschreibung und der Zeichenketten-Name werden genauso angegeben, wie bei einfachen Variablen beschrieben.
WICHTIG: Anders als bei Zahlwert-Feld-Variablen darf eine Zeichenketten-Feld-Variable nicht den gleichen Namen wie eine einfache Zeichenketten-Variable haben.

Beschreibung

Jede Zeichenketten-Variable, einfach oder indiziert, kann durch eine Teilketten-Beschreibung qualifiziert werden. Die Teilketten-Beschreibung wird verwendet, um einen Teil der Zeichenkette auszuwählen.

Wenn die Teilkette durch einen einzigen Ausdruck beschrieben ist, ist der Rest der ganzen Zeichenkette ab der durch den Ausdruck bestimmten Position gemeint.
Wenn zwei durch Kommas getrennte Ausdrücke angegeben sind, besteht die Teilkette aus den Zeichen von der durch den ersten Ausdruck bestimmten bis zu der durch den zweiten Ausdruck bestimmten Position. Der zweite Ausdruck kann kleiner als der erste sein; die Teilkette ist dann leer.
Wenn A$ eine einfache Variable ist, dann ist:

A$(3,5) die Zeichenkette mit dem dritten, vierten und fünften Zeichen aus A

A$(3,2) die leere Zeichenkette

A$ die ganze Zeichenkette

Wenn B$ eine Feld-Variable ist, dann ist:

B$(3) die ganze dritte Zeichenkette des Zeichenketten-Feldes

B$(2,3,5) das dritte bis fünfte Zeichen der zweiten Zeichenkette des Zeichenketten-Feldes

Eine Zeichenketten-Feld-Variable darf nur indiziert auftreten.
Index und Teilketten-Beschreibung können beliebige ganzzahlige Ausdrücke sein. Angenommen, die Variablen I und J werden benutzt, wobei I=5 und J=10 ist, dann ist:

C$(I) die Teilkette vom fünften bis zum letzten Zeichen von C$, wenn C$ eine einfache Zeichenketten-Variable ist, das ganze fünfte Element von C$, wenn C$ eine Zeichenketten-Feld-Variable ist.

C$(I,J) die Teilkette vom fünften bis zum zehnten Zeichen von C$, wenn C$ eine einfache Zeichenketten-Variable ist, die Teilkette vom zehnten bis zum letzten Zeichen des fünften Elementes von C$, wenn C$ eine Zeichenketten-Feld-Variable ist.

Wenn eine Teilkette sich über die logische Länge der Zeichenketten-Variablen hinaus erstreckt, werden nur die tatsächlich existierenden Zeichen geliefert und eine Warnung ausgegeben.

Beispiele

```
 10 DIM A$[10]
 20 A$="ABCDEFGHIJ"
 30 PRINT "ZEICHENKETTE A$=";A$
 40 PRINT "TEILKETTE A$(3)=";A$(3)
 50 PRINT "TEILKETTE A$(4,7)=";A$(4,7)
 60 PRINT "A$(7,5)";A$(7,5);"=LEERE ZEICHENKETTE"
*RUN
ZEICHENKETTE A$=ABCDEFGHIJ
TEILKETTE A$(3)=CDEFGHIJ
TEILKETTE A$(4,7)=DEFG
A$(7,5)=LEERE ZEICHENKETTE
```

8.3. DIM Anweisung mit Zeichenketten

Zeichenketten können als Wert einfachen oder indizierten Zeichenketten-Variablen zugewiesen werden:
Einfache Zeichenketten-Variablen und Zeichenketten-Feld-Variablen müssen in DIM Anweisungen dimensioniert werden. Der Zweck der DIM Anweisung besteht in der Reservierung von Speicher für Zeichenketten und Felder und in der Festlegung ihrer Namen und maximalen Größe.

Format

Die DIM Anweisung besteht aus dem Wort DIM gefolgt von einer Liste von Variablen- und Feld-Definition, getrennt durch Kommas.

DIM variable[größe],...,variable[größe],...

wobei variable der Name einer einfachen Zeichenketten-Variablen ist. Die besteht aus einem Buchstaben und "$" oder einem Buchstaben, einer Ziffer und "$". Die Angabe "größe" ist ein ganzzahliger Ausdruck, der die maximale Zeichenzahl

angibt, die die Zeichenkette enthalten kann.
DIM variable[dimension,größe],...,variable[dimension,größe],..
Die Dimensionsangabe bestimmt die Gesamtzahl von Elementen im Feld, "größe" gibt die maximale Zeichenzahl für jedes Element an. Nur eindimensionale Zeichenketten-Felder sind erlaubt. Sowohl dimension als auch größe sind ganzzahlige Ausdrücke.
Wenn mehr als eine Variable in einer einzigen DIM-Anweisung deklariert werden, dann müssen sie durch Kommas getrennt sein. Einfache und indizierte Zeichenketten-Variablen sowie Zahlwert-Felder (Abschnitt 7.1.) können in derselben DIM Anweisung dimensioniert werden.
Beschreibung
Zeichenketten-Felder müssen mit DIM deklariert werden; es gibt keine implizite Größe für Zeichenketten-Felder, wie das bei Zahlwert-Feldern gilt. Zeichenketten-Variablen und Elemente von Zeichenketten-Feldern werden mit der leeren Zeichenkette initialisiert.
WICHTIG: Zeichenketten-Variablen und -Felder können nicht redimensioniert werden.
Beispiel

```
 10 DIM A$[28],B$[4,20],C$[4,3]
 20 LET A$="DER TITEL VON ABSCHNITT IST"
 30 FOR K=1 TO 4
 40   READ B$[K]
 50 NEXT K
 60 FOR K=1 TO 4
 70   READ C$[K]
 80 NEXT K
 90 DATA "EINFUEHRUNG"
100 DATA "ALLGEMEINES"
110 DATA "KOMMANDOS"
120 DATA "GRUNDELEMENTE"
130 DATA " 1","2","3","4"
```

```
140 FOR K=1 TO 4
150   PRINT A$[1,24];C$[K];A$[24],B$[K]
160 NEXT K
=RUN

DER TITEL VON ABSCHNITT 1 IST EINFUEHRUNG
DER TITEL VON ABSCHNITT 2 IST ALLGEMEINES
DER TITEL VON ABSCHNITT 3 IST KOMMANDOS
DER TITEL VON ABSCHNITT 4 IST GRUNDELEMENTE
```

Eine Teilkette von A$ wird gedruckt, gefolgt vom K-ten Element von C$, einer weiteren Teilkette von A$ und dem K-ten Element von B$. Dieses Beispiel listet die Titel der ersten vier Abschnitte dieser Dokumentation.

8.4. Zeichenketten-Ausdrücke

Zeichenketten-Ausdrücke bestehen aus einem oder mehr Quell-Ketten (Zeichenketten-Konstanten, -Variablen, -Funktionen) kombiniert von links nach rechts durch den Konkatenations-(Verkettungs-) Operator (+). Es entsteht so ein neuer Zeichenketten-Wert. Zeichenketten-Ausdrücke können an Zeichenketten-Variablen zugewiesen oder mit anderen Zeichenketten-Ausdrücken zur Bildung eines Zahlwertes (0 oder 1) verglichen werden.

Format

Das Format ist eine Liste von Quell-Ketten getrennt durch "+".

```
zeichenkette
zeichenkette + zeichenkette...
```

Jede Quell-Kette kann eine Zeichenketten-Konstante, -Variable oder -Funktion sein.

Beschreibung

Eine Quell-Kette ist eine beliebige Einheit, die eine Zeichenkette ergibt. Der Wert einer Quell-Kette ist wie unter "Zeichenketten-Konstanten", "Zeichenketten-Variablen" und "Zeichenketten-Funktionen" beschrieben.

Ein Beispiel für eine Zeichenketten-Konstante ist "BASIC" oder "<10>"; für eine Zeichenketten-Variable ist C5$(2), B$(2,3) oder A$(5,3,10); für eine Zeichenketten-Funktion ist CHR$(208) ein Beispiel.
Das "+"-Zeichen, zwischen zwei Quell-Ketten benutzt, ist der Konkatenations-Operator. Die Konkatenation zweier Zeichenketten ergibt eine vorübergehend angelegte Zeichenkette, deren Zeichen diejenigen der ersten Kette, direkt gefolgt von denen der zweiten Kette sind. Diese vorübergehend angelegte Zeichenkette kann für weitere Konkatenations-Operationen oder für Zeichenketten-Vergleiche verwendet werden oder sie kann einer Zeichenketten-Variablen als Wert zugewiesen werden.
Die maximale Länge für jede vorübergehend angelegte Zeichenkette ist 32767 Zeichen. Die Eingangs-Operanden werden durch die Konkatenation nicht berührt.

Erlaubte Zeichenketten-Ausdrücke sind:

```
A$+B$(2)+"<10><13>ABCD"+C$(3,1,2)
"BASIC" +C5$(2)
"BASIC"
C5$(2)
```

Beispiel

```
10 DIM A$[5],B$[10,10]
20 LET A$="VER",B$="KETTE"
30 PRINT A$+B$[1,1,4]+"UNG"
*RUN
VERKETTUNG
```

8.5. Zeichenkettenbezogene Funktionen

Es gibt eine Zahl von vordefinierten Funktionen in SIOS BASIC, die Zeichenkettenwerte als Parameter akzeptieren und/oder einen Zeichenkettenwert als Ergebnis liefern. (Benutzer-definierte Zeichenketten-Funktionen werden in Abschnitt 9 beschrieben.)

8.5.1. CHR$ Funktion

CHR$(ganzzahliger ausdruck)

wobei ein ganzzahliger ausdruck einen Wert zwischen 0 und 255 inklusive ergeben muß. Der Wert von CHR$ ist das Zeichen, das zum Wert des Ausdrucks gemäß dem Standard-Zeichensatz (Anhang A) gehört. Als Beispiel:

```
 10 PRINT CHR$(65)
*RUN
A
```

8.5.2. ASC Funktion

ASC(zeichenketten-ausdruck)

ASC liefert den Zahlwert des ersten Zeichens der als Parameter übergebenen Zeichenkette gemäß dem Standard-Zeichensatz (AnhangA). Als Beispiel:

```
 10 PRINT ASC("A")
*RUN
 65
```

8.5.3. LEN Funktion

LEN(zeichenketten-ausdruck)

LEN liefert die logische Länge der als Parameter übergebenen Zeichenkette. Als Beispiel:

```
 10 DIM A$[20]
 20 LET A$="ABCDEFG$"
 30 PRINT LEN(A$)
*RUN
 8
```

8.5.4. POS Funktion

POS(zeichenketteA,zeichenketteB)

wobei zeichenketteA und zeichenketteB beliebige Zeichenkettenausdrücke sind. POS gibt die kleinste ganze Zahl zurück, die die Startposition einer Teilkette von zeichenketteA angibt, die genau der zeichenketteB gleich ist. Wenn zeichenketteB keine Teilkette von zeichenketteA ist, wird der Wert Null zurückgegeben. Wenn zeichenketteB die Länge Null hat, wird POS den Wert 1 zurückgeben. Beispiel:

```
10 PRINT POS("12ABC34" ,"C3")
*RUN
 5
```

8.5.5. VAL Funktion

VAL(zeichenkettenausdruck)

VAL liefert den Zahlwert, der durch die Zeichen in zeichenkettenausdruck dargestellt ist. Es ergibt einen Fehler, wenn am Anfang der Kette keine erlaubte Zahldarstellung gefunden wird. Es wird angenommen, daß die Zahl mit dem ersten Zeichen des Zeichenkettenausdrucks beginnt und vor dem ersten Zeichen aufhört, das nicht zu einer gültigen Zahldarstellung gehören kann. Leerzeichen werden ignoriert und E-Schreibweisen können benutzt werden.
Beispiel:

```
10 DIM A$[30]
20 LET A$="123X4EZ"
30 PRINT VAL(A$)
40 PRINT VAL(A$[5,6]+"9")
*RUN
 123
 4000000000
```

8.5.6. STR$ Funktion

STR$(zahlwertausdruck)

Die STR$ Funktion ergibt die Zeichenkettendarstellung des

Parameters. Die Zeichenkette ist die, die durch eine PRINT Anweisung produziert wurde, mit der Ausnahme, daß Leerzeichen entfernt werden. Beispiel:

```
10 DIM A$[4761]
20 LET A$=STR$(2.36x4)
30 PRINT A$;
40 PRINT VAL(A$)
50 PRINT STR$(VAL(A$))
*RUN
9.44
 9.44
9.44
```

8.5.7. LEFT$ Funktion

LEFT$(zeichenkettenausdruck,ganzzahliger ausdruck)

LEFT$ gibt die n ersten (linkesten) Zeichen des Zeichenkettenausdrucks zurück. Der ganzzahlige Ausdruck gibt die Position des letzten zurückgegebenen Zeichens an.
Beispiel:

```
10 PRINT LEFT$("ABCDE" ,3)
*RUN
ABC
```

8.5.8. RIGHT$ Funktion

RIGHT$(zeichenkettenausdruck,ganzzahliger ausdruck)

RIGHT$ gibt die letzten (rechtesten) Zeichen des Zeichenkettenausdrucks zurück, vom n-ten Zeichen bis zum Ende. Das n-te Zeichen wird durch den Wert des ganzzahligen Ausdrucks bestimmt. Beispiel:

```
10 PRINT RIGHT$("ABCDEFGHIJ" ,4)
*RUN
DEFGHIJ
```

8.5.9. SEG$ Funktion

SEG$(zeichenkettenausdruck,ganzzahliger ausdruck,
ganzzahliger ausdruck)

SEG$ gibt die Teilkette von Zeichen des Zeichenkettenausdrucks zurück, vom durch den ersten ganzzahligen Ausdruck bezeichneten bis zum durch den zweiten ganzzahligen Ausdruck bezeichneten Zeichen. Beispiel:

```
10 PRINT SEG$("ABCDEFG" ,3,6)
*RUN
CDEF
```

8.6. Vergleich von Zeichenketten

Zeichenketten-Ausdrücke können durch Vergleichs-Operatoren verglichen werden. Das Ergebnis ist "wahr" (Zahl 1), wenn der Vergleich stimmt, oder "falsch" (Zahl 0), wenn der Vergleich nicht stimmt.

=	Gleich
<>	Ungleich
<	Kleiner als
>	Größer als
<=	Kleiner oder Gleich
>=	Größer oder Gleich

Zwei Zeichenketten sind nur dann gleich, wenn sie die gleiche Länge haben und in jedem Zeichen übereinstimmen. Eine Zeichenkette ist kleiner als eine andere, wenn der Zahlwert des ersten Zeichens, das von entsprechenden Zeichen in der zweiten Zeichenkette abweicht, kleiner ist als der Zahlwert des Zeichens in der zweiten Kette (der Zahlwert richtet sich nach dem Standard-Zeichensatz in Anhang A) oder wenn sie ein echter Anfang der zweiten Zeichenkette ist ("AB"<"ABC").

Ein Zeichenketten-Vergleich kann in jedem Zahlwert-Ausdruck vorkommen, weil das Ergebnis eine Zahl ist. Die Zeichenket-

ten-Vergleichs-Operatoren haben denselben Rang in der Operator-Hierarchie wie die Vergleichs-Operatoren für Zahlwerte. Beispiele für Zeichenketten-Vergleiche:

```
A$=B$
A$=B$!C$ =D$
(A$<>"BOB")+5
```

In Abschnitt 4.6. findet sich eine Beschreibung der Vergleichs-Operatoren und die Hierarchie.
Zeichenketten-Vergleiche werden üblicherweise in IF-Anweisungen verwendet.

```
 10 DIM A$[10],B$[10]
 15 FOR K=1 TO 4
 20 READ A$,B$
 30 IF A$<B$ THEN PRINT A$;"<60>";B$
 40 ELSE DO
 50   IF A$=B$ THEN PRINT A$;"=";B$
 60   ELSE PRINT A$;">";B$
 70 DOEND
 80 NEXT K
 90 DATA "ABC","ABCD" , "ABC","B"
100 DATA "ABC","ABC" ,"C",""
*RUN
ABC<ABCD
ABC<B
ABC=ABC
C>
```

8.7. Zeichenkettenzuweisung

Der Zuweisungs-Operator (=) kann benutzt werden, um einen Zeichenketten-Wert (definiert durch einen Zeichenketten-Ausdruck) an eine oder mehrere Zeichenketten-Variablen (oder Teilkette von Zeichenketten-Variablen) zuzuweisen. Mehrere verschiedene Zuweisungen können in einer LET Anweisung enthalten sein.

Format

LET variable=ausdruck

LET variable=ausdruck,...,variable=ausdruck

Das Wort LET ist völlig optional und kann entfallen. Die Variable ist eine ganze Zeichenketten-Variable (einfach oder indiziert) oder Teil einer Zeichenketten-Variablen (angezeigt durch Teilketten-Beschreibung) in die der Zeichenketten-Wert kopiert wird. Zahlwert-Zuweisungen sind in Abschnitt5.1. beschrieben und können mit Zeichenketten-Zuweisungen in derselben LET Anweisung gemischt werden.

Beschreibung

Die Ausführung einer LET Anweisung geschieht wie folgt. Die Indizes der Variablen, an die zugewiesen wird, werden von links nach rechts ausgewertet. Danach wird der Ausdruck ausgewertet und der Variablen zugewiesen. Die Art und Weise, in der sich jede Zuweisung auswirkt, hängt von der Art der Teilketten-Beschreibung ab, die bei der Zielvariablen angegeben ist.

Wenn es keine Teilketten-Beschreibung gibt, wird der Wert der ganzen Variablen durch den Zeichenketten-Wert ersetzt. Wenn der neue Wert ganz in die Variable paßt, wird die logische Länge der Variable auf den Wert der Länge des neuen Werts gesetzt. Wenn die Variable zu klein ist, wird eine Warnung gedruckt, der Wert wird rechts abgeschnitten und die logische Länge wird gleich der physischen Länge.

Wenn als Teilketten-Beschreibung ein Wert angegeben ist, gibt dieser die Anfangsposition für die Zuweisung an. Der ganze Zeichenketten-Wert wird in die Variable kopiert, beginnend bei der angegebenen Position und fortfahrend bis zum physischen Ende der Variablen oder dem Ende der Zeichenkette, je nachdem, was zuerst kommt. Der Teil der Variablen, der vor der Anfangsposition liegt, bleibt unverändert. Die Anfangsposition darf nicht mehr als um 1 größer sein als die aktuelle logische Länge der Variablen (d.h., es dürfen keine undefinierten Zeichen-Positionen mitten in der Zeichenket-

ten-Variablen auftreten). Wenn die Variable zu klein ist, wird der Wert rechts abgeschnitten und eine Warnung ausgegeben.
Wenn als Teilketten-Beschreibung zwei Werte angegeben sind, wird dadurch ein Teilfeld der Variablen bestimmt, in das der Zeichenketten-Wert gebracht werden soll. Wenn nötig, wird der Wert rechts abgeschnitten und eine Warnung ausgegeben oder es werden Leerzeichen zur Ergänzung eingefügt, falls der Wert zu kurz ist. Die Teilkette, die den Wert aufnehmen soll, muß innerhalb der physischen Grenzen der Zeichenketten-Variablen liegen; ebneso müssen alle oben erwähnten Regeln eingehalten werden. Die neue logische Länge der Variablen ist der größere Wert aus der alten logischen Länge und der letzten Position der Teilkette. Jedes Zeichen des alten Wertes, das links oder rechts von der Teilkette liegt, bleibt unverändert.

Beispiel

```
 10 DIM A$[10]
 20 LET A$="1234567890"
 30 PRINT A$
 40 LET A$[5]="ABCDEF"
 50 PRINT A$
 60 LET A$[7,9]="1234"
 70 PRINT A$
 80 LET A$[6,8]="X"
 90 PRINT A$
100 LET A$=A$[1,4]+"567890"
110 PRINT A$
*RUN
1234567890
1234ABCDEF
```

```
WARNING 146 AT   60
1234AB123F
1234AX  3F
1234567890
```

In Zeile 60 wird die Zeichenkette "1234" auf 3 Zeichen gekürzt ("123"), um in die Teilkette A$(7,9) zu passen.
In Zeile 80 wird die Teilkette A$(6,8) mit Leerzeichen aufgefüllt, da "X" nur ein Zeichen hat. Der Endwert von A$ ist derselbe wie der ihr in Zeile 20 ursprünglich zugewiesene Wert.
Das Beispiel illustriert verschiedene Möglichkeiten der Zuweisung an Teilketten von Feldelementen:

```
10 DIM A$[3,5]
20 A$[1]="ABCDE" ,A$[2]="ABCDE",A$[3]="ABCDE"
30 LET A$[1,3]=A$[2]
40 PRINT A$[1],A$[2],A$[3]
50 LET A$[2,4,5]=A$[3]
60 PRINT A$[1],A$[2],A$[3]
70 LET A$[2]=A$[1,1,1],A$[3,2,3]=A$[1,1,1]
80 PRINT A$[1],A$[2],A$[3]
*RUN

WARNING 146 AT   30
ABABC            ABCDE           ABCDE

WARNING 146 AT   50
ABABC            ABCAB           ABCDE
ABABC            A               AA DE
```

8.8. Zeichenketteneingabe-Zuweisung

Die INPUT Anweisung kann zur Zuweisung von Zeichenketten an Zeichenketten-Variablen durch Eingabe über die Tastatur benutzt werden.
Zeichenketten können dabei mit oder ohne Anführungszeichen eingegeben werden. Wenn ohne, werden Leerzeichen am Anfang

und am Ende entfernt und das Eingabe-Element endet bei einem Komma oder durch Drücken der ET 1 - Taste.
Die Regeln zur Zuweisung eines Wertes an eine Variable sind die unter "Zeichenketten-Zuweisung" (Abschnitt 8.7) beschriebenen.

Beispiele

```
 10 DIM A$[16],B$[2,5],C$[40]
 20 INPUT A$,B$[1],B$[2],C$
 25 PRINT
 30 PRINT A$;B$[1];B$[2];C$
*RUN
?"DER WERT VON B$=","1234 "," 2X5 ", "X5=ABC"
DER WERT VON B$=1234 2X5 X5=ABC
```

8.9. Zeichenketteneingabe durch LINPUT Anweisung

Die LINPUT Anweisung akzeptiert die Zeichen, die der Anwender über die Tastatur eingibt und weist sie als Zeichenkette einer angegebenen Zeichenketten-Variablen zu.

Format

LINPUT zeichenketten-variable

LINPUT zeichenketten-konstante,zeichenketten-variable

wobei die Zeichenketten-Variable das Ziel der Eingabe ist. Die Variable kann einfach oder indiziert sein. In der zweiten Form ersetzt die Zeichenketten-Konstante das übliche Bereitschaftszeichen "?" (Fragezeichen).

Beschreibung

Alle Zeichen einschließlich Anführungszeichen, Kommas und Leerzeichen werden akzeptiert. Abgeschlossen wird die Eingabe durch das Wagenrücklauf-Zeichen (CR).

Beispiel

```
 10 DIM A$[20]
 20 PRINT "TIPPE 20 ZEICHEN:"
 30 LINPUT "",A$
 35 PRINT
 40 PRINT A$
 50 LINPUT "TIPPE 5 ZEICHEN:";A$
 55 PRINT
 60 PRINT A$
*RUN
TIPPE 20 ZEICHEN:
"BELIEBIGE ZEICHEN".
"BELIEBIGE ZEICHEN"
TIPPE 5 ZEICHEN:E"+"x
E"+"x
```

Weil mehr als 20 Zeichen (Länge von A$) in Zeile 30 vom Benutzer eingetippt wurden, wurde der Punkt bei der ersten Eingabe abgeschnitten. Bei beiden Eingaben wurden Anführungszeichen als Bestandteil des Textes behandelt.

8.10. Zeichenkettenausgabe durch PRINT Anweisung

Jeder Zeichenketten-Ausdruck kann auf das Ausgabegerät mittels der PRINT Anweisung ausgegeben werden. Die Größe des Ausgabe-Feldes ist die Zahl der Druckzeichen in der Zeichenkette. Wenn dem Zeichenketten-Ausdruck ein Komma vorausgeht, wird es in den nächsten Abschnitt der Ausgabezeile gedruckt. Jede Druckzeile ist in fünf Abschnitten zu je 14 Zeichen unterteilt (siehe PRINT Anweisung, Abschnitt 5.8.). Wenn dem Zeichenketten-Ausdruck ein Semikolon vorausgeht, wird er unmittelbar in Anschluß an den vorausgehenden Output ausgegeben.
Zeichenketten können über den Bildschirm bzw. Drucker in speziellen Formaten mittels der PRINT USING Anweisung (siehe Abschnitt 11. Formatierte Ausgabe) ausgegeben werden. Auf Da-

teien können Zeichenketten wie in Abschnitt 10 beschrieben ausgegeben werden.

Beispiel

```
  10 DIM C$[10],N5$[3,5]
  20 LET C$="XK9-753-20",A=2.5,B=1E-19,N5$[1]="ABCDE"
  30 PRINT A,B,C$
  40 PRINT "BOB"+C$,N5$[1]
  50 PRINT C$+"BOB";N5$[1]
  60 PRINT "<10><34>LINE<34><10><13>-1"
*RUN
 2.5            1.00000E-19           XK9-753-20
BOBXK9-753-20  ABCDE
XK9-753-20BOBABCDE

"LINE"
-1
```

In Zeile 60 bewirkt das Zeichen <10> (LF) einen Zeilenvorschub und das Zeichen <13> (CR) einen Wagenrücklauf. Das Zeichen <34> (") bewirkt die Ausgabe eines Anführungszeichens. Die eigentlichen Anführungszeichen (") vor und nach der Zeichenkette in Zeile 60 werden nicht gedruckt.

8.11. Zeichenketten in READ/DATA/RESTORE Anweisungen

READ, DATA und RESTORE Anweisungen können auch in Verbindung mit Zeichenketten-Variablen, die einfach oder indiziert, mit oder ohne Teilketten-Beschreibung sind, benutzt werden. Die Zeichenketten-Variable ist in der READ Anweisung aufgeführt und in der DATA Anweisung muß an entsprechender Stelle eine Zeichenkette auftreten. Eine RESTORE Anweisung kann benutzt werden, um das wiederholte Lesen einer DATA Anweisung durch nachfolgende READ Anweisungen zu ermöglichen. Die vollständige Beschreibung von READ/DATA/RESTORE ist in Abschnitt 5.9. zu finden.

Zeichenketten-Variablen können mit Zahlwert-Variablen in READ Anweisungen gemischt auftreten, aber die entsprechenden Konstanten für Zahlwert-Variablen müssen Zahlwerte sein. Eine Zeichenketten-Konstante wird der Zeichenketten-Variable gemäß den Regeln für Zeichenketten-Zuweisung in diesem Abschnitt zugewiesen. In Zeichenketten-Variablen können sowohl Zeichenketten als auch Zahlwerte aus DATA Anweisungen eingelesen werden. In jedem Fall wird die Zeichendarstellung wie sie in DATA auftritt, als Wert benutzt.
Zeichenketten können auch von Dateien gelesen werden (siehe Abschnitt 10.).

Beispiel

```
 10 DIM A$[20],B$[20]
 20 DATA "BOB","<10>JONES"
 30 READ A$[1,3]
 40 READ A$[4,9]
 50 LET B$="HI"
 60 PRINT B$,A$
*RUN
HI       BOB
            JONES
```

Wenn die PRINT Anweisung ausgeführt wird, wird ein Zeilenvorschub (<10>,LF) gedruckt.

9. Benutzerdefinierte Funktionen

Eine benutzerdefinierte Funktion ist eine, die innerhalb des Anwenderprogramms definiert ist und auch von dort auf die gleiche Weise wie eine eingebaute Funktion aufgerufen wird. Funktionsnamen bestehen aus dem Buchstaben "FN" und einem normalen Variablennamen. Der Typ der Funktion ist der durch den Aufbau des Namens bestimmte Typ. Die Funktion gibt einen INTEGER Wert zurück, wenn der Name mit "%" endet, Zeichen-

kette, wenn er mit "$" endet, REAL sonst.
Eine Funktion wird innerhalb eines Ausdrucks durch Angabe des Namens und gegebenfalls einer geklammerten Liste von Parameter-Werten aufgerufen. Der von der Funktion zurückgegebene Wert nimmt die Stelle des Funktionsaufrufs ein.

Es gibt zwei Komplexitäts-Ebenen bei der Definition einer SIOS BASIC Funktion. Auf der einfachen Ebene werden mit einer einzeiligen Funktion ein Funktionsname und eine Parameterliste mit einem Ausdruck gleichgesetzt, der die Parameter zur Berechnung des Ergebniswertes verwenden kann. Die mehrzeilige Funktion ist eine komplexere Einheit; sie kann aus vielen Anweisungen bestehen. Sie gibt ihr Ergebnis mit einer RETURN Anweisung zurück.
Einige Hinweise zu den in SIOS-BASIC vorhandenen Funktionen finden sich im Abschnitt4.4., Funktionen. Eine komplette Liste der eingebauten Funktionen, die dem Benutzer von BASIC zur Verfügung stehen, findet sich in Anhang D.

9.1. Einzeilige Funktionen

Eine einzeilige Funktion ist in einer Zeile vollständig definiert, wozu eine DEF Anweisung benutzt wird; das Ergebnis wird durch einen Ausdruck berechnet.

Format

Die Formate für einzeilige Funktions-Definitionen sind:

DEF funktionsname(formale parameterliste)=ausdruck

DEF funktionsname=ausdruck

Sowohl Funktionsname als auch Elemente der formalen Parameter-Liste können jeden Typ haben:

INTEGER (sie enden auf "%")

Zeichenkette (sie enden auf "$")

REAL (sonst)

Der Typ des Ausdrucks muß mit dem Typ des Funktionsnamen vereinbar sein. Der Ausdruck kann ein beliebiger, gültiger Zahlwert- oder Zeichenketten-Ausdruck sein und kann sowohl von Parametern als auch von Programm-Variablen Gebrauch machen.

Beschreibung

Die Parameter in einer Funktions-Definition sind formale Parameter; wenn die Funktion aufgerufen wird, werden sie durch die aktuellen Parameter ersetzt, die an die Funktion übergeben werden. Alle Variablen, die als formale Parameter benntzt werden, sind in der Funktion lokal; d.h. sie stehen in keiner Beziehung zu irgendwelchen Programm-Variablen mit gleichem Namen. Die formalen und aktuellen Parameter werden paarweise aufeinander abgebildet, je nach der Position, in der sie in der Liste auftreten.
Die DEF Anweisung ist ausführbar, obwohl die dadurch definierte Funktion nur durch Ansprechen innerhalb eines Ausdrucks aufgerufen werden kann. Die DEF Anweisung für eine Funktion muß ausgeführt werden, bevor die Funktion aufgerufen wird.
Durch aufeinanderfolgende DEF Anweisungen, die dieselbe Funktion (gleicher Name) definieren, wird diese Funktion umdefiniert. Die jeweils ältere Definition wird vergessen.

Beispiele

```
10 DEF FNZ(C,D)=Cx(D+10)-6
```

Die Funktion FNZ ist vom Typ REAL. Die Formal-Parameter C und D sind ebenfalls reell. Beim Aufruf werden C und D Werte zugewiesen; dann wird der Ausdruck Cx (D+10)-6 ausgewertet und das Ergebnis ersetzt den Funktions-Namen, wo er im Ausdruck auftritt.

```
20 DEF FNG$(K$,L$)=K$+L$+K$
```

Die Funktion FNG$ ist eine Zeichenketten-Funktion. Die formalen Parameter K$ und L$ sind Zeichenketten-Variablen und

erhalten die Werte der beim Aufruf verwendeten aktuellen Parameter. Beim Aufruf wird die Verkettung der Werte K$,L$ und K$ als Ergebnis ermittelt. Dieses Ergebnis ersetzt den Funktions-Namen, wo er im Ausdruck auftritt.

```
30  DEF FNB%(A%,X2%)=A%xX2%+(A%+X2%)
```

Die Funktion FNB% ist eine ganzzahlige Funktion, die bei Aufruf einen ganzzahligen Wert liefert. Die Berechnungen werden mit ganzzahliger Arithmetik ausgeführt, da sowohl A% als auch X2% ganzzahlig sind.

9.2. Mehrzeilige Funktionen

Eine mehrzeilige Funktion wird in Form von mehreren aufeinanderfolgenden Anweisungen geschrieben. Die erste Anweisung ist eine DEF Anweisung, die letzte eine FNEND Anweisung. Die Ausführung der Funktion endet bei Auftreten einer RETURN Anweisung; dann wird der Ergebniswert an die Stelle des Aufrufs geliefert.

Format

Eine mehrzeilige Funktions-Definition hat drei Teile: den Funktions-Kopf, den Funktions-Rumpf und das Funktions-Ende.
Der Funktionskopf hat eine der Formen:

```
DEF  funktionsname(formale parameterliste)
DEF  funktionsname
```

Alle Teile einer solchen Anweisung sind bereits in Abschnitt 9.1. für einzeilige Funktionen beschrieben.
Der Funktionsrumpf besteht aus einer Folge von Anweisungen, einschließlich mindestens einer RETURN Anweisung:

```
RETURN ausdruck
```

Der Ausdruck hat je nach dem Typ der Funktion Zahlwert oder ist Zeichenkette. Bei Zahlwert-Funktionen wird der RETURN Ausdruck in den Typ der Funktion umgewandelt.

Das Funktionsende besteht aus folgender Anweisung:

FNEND

Diese Anweisung muß immer die letzte Anweisung in der Funktions-Definition sein.

Beschreibung

Der Rumpf einer Funktion kann beliebige SIOS BASIC Anweisungen, mit folgenden Einschränkungen, enthalten:

1) Innerhalb des Funktions-Rumpfes darf keine Funktions-Definition auftreten. Funktions-Aufrufe sind jedoch erlaubt, eingeschlossen Aufrufe derselben Funktion.

2) Der Funktions-Rumpf muß abgeschlossen sein. FOR Schleifen und DO Blöcke müssen vollständig innerhalb des Rumpfes liegen und Sprünge dürfen nicht von innerhalb des Rumpfes nach außerhalb gehen und umgekehrt.

Die formalen Parameter im Kopf einer mehrzeiligen Funktion werden in der gleichen Weise benutzt wie bei einzeiligen Funktionen. Die formalen Parameter können im Funktions-Rumpf geändert werden. Der Wert der aktuellen Parameter wird durch Änderungen der formalen Parameter jedoch niemals beeinflußt.
Die folgende mehrzeilige Funktion liefert einen Zeichenketten-Wert, die formalen Parameter sind Zeichenketten-Variablen:

```
10  DEF FNR$(A$)
20    REM..FNR$ LIEFERT DAS GESPIEGELTE A$
30    IF LEN(A$)<=1 THEN RETURN A$
40    RETURN FNR$(A$[2])+A$[1,1]
50  FNEND
```

9.3. Aufruf einer benutzerdefinierten Funktion

Eine benutzerdefinierte Funktion wird durch Verwendung des Funktionsnamens innerhalb eines Ausdrucks aufgerufen; gegebenfalls mit geklammerter aktueller Parameter-Liste. Der Funktions-Aufruf wird durch den zurückgegebenen Funktionswert ersetzt.

Format

Ein Funktions-Aufruf hat die Form:

funktionsname(aktuelle parameterliste)

funktionsname

Die optionale Parameter-Liste enthält einen oder mehrere durch Komma getrennte aktuelle Parameter. Ein aktueller Parameter kann ein Zahlwert-Ausdruck oder ein Zeichenketten-Ausdruck sein:

Beschreibung

Aktuelle Parameter können nur benutzt werden, um einmalig Werte an eine Funktion zu übergeben. Diese werden gewöhnlich innerhalb der Funktion benutzt, obwohl das nicht erforderlich ist.
Die Zahl der aktuellen Parameter im Funktions-Aufruf muß mit der Zahl der formalen Parameter in der Funktions-Definition übereinstimmen. Die Namen der entsprechenden Parameter müssen nicht die gleichen sein. Aktuelle und formale Parameter werden entsprechend ihrer Position in der jeweiligen Liste aufeinander abgebildet. Zum Beispiel entspricht der dritte aktuelle Parameter in einem Funktions-Aufruf dem dritten formalen Parameter in der DEF Anweisung.
Wenn der formale Parameter ein einfacher Zahlwert ist (V), dann kann der aktuelle Parameter ein Zahlwert-Ausdruck sein, der einen einfachen Zahlwert ergibt, oder eine einfache oder indizierte Zahlwert-Variable (2xV,V,5x7,V(5)). Wenn die Variablen unterschiedlichen Typ haben oder der aktuelle Para-

meter ein Ausdruck ist, werden notwendige Typumwandlungen durchgeführt, wie in Abschnitt 6.5., Zahlwert-Zuweisung beschrieben.
Wenn der formale Parameter eine einfache Zeichenketten-Variable ist, muß der entsprechende aktuelle Parameter ein Zeichenketten-Ausdruck sein.

Beispiele

Beim Aufruf der einzeiligen Funktion

```
10  DEF FNZ(C,D)=Cx(D+10)-6
```

sind die aktuellen Parameter Zahlwert-Variablen vom gleichen Typ:

```
500 LET C=5,D=2
510 PRINT FNZ(C,D)
*RUN
 54
```

Die aktuellen Parameter können auch Zahlwert-Ausdrücke sein:

```
520 PRINT FNZ(5,2)
*RUN
 54
```

Beim Aufruf der Zeichenketten-Funktion

```
20  DEF FNG$(K$,L$)=K$+L$+K$
```

können die aktuellen Parameter Zeichenketten-Variablen sein:

```
530 K$="ABC",L$="123"
540 PRINT FNG$(K$,L$)
*RUN
ABC 123ABC
```

oder Zeichenketten-Ausdrücke:

```
550 PRINT FNG$("ABC" , "123")
```

```
*RUN
ABC123ABC
```

Beim Aufruf der Funktion:

```
30  DEF FNB%(A%,X2%)=A%xX2%+(A%+X2%)
```

die einen ganzzahligen Wert liefert, können die aktuellen Parameter Variablen sein:

```
500 LET X%=4,Y%=2
510 PRINT FNB%(X%,Y%)
*RUN
 14
```

oder Zahlwert-Ausdrücke:

```
520 PRINT FNB%(4,2)
*RUN
 14
```

Jedes der oben gezeigten Beispiele ist eine einzeilige Funktion, die einen einfachen Wert liefert. Die formalen Parameter werden durch die Ausführung der Funktion nicht beeinflußt. In einer mehrzeiligen Funktion können die formalen Parameter im Funktions-Rumpf geändert werden. Der Wert der aktuellen Parameter wird jedoch niemals durch Änderung der formalen Parameter beeinflußt.
Die folgende mehrzeilige Funktion liefert einen Zeichenketten-Wert, der das Spiegelbild des Zeichenketten-Wertes ist, der über den aktuellen Parametern eingegeben wird:

```
10  DEF FNR$(A$)
20    REM..FNR$ LIEFERT DAS GESPIEGELTE A$
30    IF LEN(A$)<=1 THEN RETURN A$
40    RETURN FNR$(A$[2])+A$[1,1]
50  FNEND
```

Bei Aufruf dieser Funktion kann der aktuelle Parameter eine Zeichenketten-Konstante sein:

```
 70 PRINT FNR$("ABCDE")
*RUN
EDCBA
```

Der aktuelle Parameter kann ebenso eine Zeichenketten-Variable sein:

```
 60 DIM X$[5]
 70 X$="12345"
 80 PRINT "FNR$ RETURNS:";FNR$(X$)
*RUN
FNR$ RETURNS:54321
```

10. Dateien

Zur Speicherung von umfangreichen Datenmengen außerhalb eines Anwenderprogrammes wird in SIOS-BASIC eine umfassende Dateiarbeit zur Verfügung gestellt. Diese erlaubt eine flexible, direkte Manipulation von großen Beständen von in Dateien gespeicherten Daten. Das Anlegen der Dateien hat mit dem entsprechenden Dienstprogramm des SIOS-Betriebssystems zu erfolgen. (FGEN)
Der Dateiname kann aus bis zu 17 signifikanten Stellen bestehen. Werden mehr Zeichen angegeben, bleiben die letzten Zeichen unberücksichtigt. Im übrigen gelten die Bedingungen für den physischen Dateinamen entsprechend Systemhandbuch - Anlage III. 3.3. (Ausgabe Mai 1980).
Die Dateien werden in einem Format gemäß der KROS Nr. 5108/01 bzw. 5110/01 aufgezeichnet (siehe dazu auch Systemhandbuch Teil III, Abschnitt 2.1. Datenformat und Datenorganisation der Diskette).
Es kann mit mehreren Dateien gleichzeitig gearbeitet werden. Alle bei SIOS 1526 zugelassenen Formate können verwendet werden.

Es werden 2 Dateitypen unterschieden: Binärdateien und Textdateien. Der Zugriff zu diesen Dateitypen erfolgt mit unterschiedlichen Anweisungen:

READ/WRITE für Binärdateien

INPUT/LINPUT/PRINT für Textdateien

In Binärdateien werden die Werte in festen Längen gelesen bzw. geschrieben. Die Speicherung der Werte erfolgt auf der Diskette im internen Format.
In Textdateien ist die Länge der Daten vom Wert abhängig. Die Speicherung der Werte erfolgt auf der Diskette im KOI-7-Bit-Code. Im allgemeinen sind Textdateien nur über BASIC verarbeitbar.

Der erste Satz der Datei wird Satz Ø genannt.
Wenn auf eine Datei zugegriffen wird, verwaltet BASIC einen Zeiger auf die Datei. Er gibt die Position des nächsten zu lesenden oder zu schreibenden Zeichens an. Der Zeiger rückt schrittweise durch die Datei, wenn sequentielle Ein- oder Ausgaben erfolgen.

10.1. Öffnen von Dateien: FILE-Anweisung

Um vom Programm aus auf eine Datei zugreifen zu können, muß sie offen sein. Für jede offene Datei wird ein Pufferbereich benötigt. Dieser Puffer ist ein zusätzlicher Bereich, der für die Verwaltung der Daten im Zusammenhang mit der Ein- und Ausgabe auf Dateien benötigt wird (siehe dazu auch Abschnitt 3.2.2. NEW-Kommando).
Für jede zu öffnende Datei wird eine logische Verbindung zwischen der in den Zugriffsanweisungen benutzten Dateinummer unddem Dateinamen hergestellt. Die Dateinummer ist eine ganze Zahl zwischen 1 und 15.

Die Verbindung zwischen Dateiname und Dateinummer wird durch die FILE Anweisung realisiert. FILE bewirkt, daß einem Dateinamen eine Dateinummer zugeordnet wird. Wenn der Dateinummer bereits eine Datei zugeordnet ist, wird diese geschlossen.

Format

Das Format für FILE ist

```
FILE #dateinummer; namen-optionen-kette
     [,return variable]
```

Die Dateinummer ist eine Zahl zwischen 1 und 15. Die Namen-Optionen-Kette ist ein Zeichenkettenausdruck. Sie enthält einen Dateinamen gefolgt von optionalen Parametern, alle durch Semikolon abgegrenzt. Die Optionen (ACC, RND) sind in Tafel 10-1 erläutert. Die zweite Form von FILE schließt die Angabe einer Return-Variablen ein, die Zahlwert haben muß. Sie wird benutzt, um den Ausführungszustand von FILE zu liefern.

Beschreibung

Der Dateiname wird der Dateinummer zugeordnet. Dann wird die Datei geöffnet. Nachfolgende Zugriffe zur angegebenen Dateinummer betreffen die benannte Datei. Die FILE Anweisung erlaubt die Angabe einiger Optionen in der Namen-Optionen-Kette. Sie und ihre Wirkung sind in Tafel 10-1 beschrieben.

Tafel 10-1

Datei-Optionen-Parameter

Parameter	Wirkung
REC= log. Satzlänge	Spezifiziert die log. Satzlänge, die Angabe muß mit der entsprechenden Eintragung im Kennsatz übereinstimmen. Der Parameter muß bei wahlfreiem Zugriff angegeben werden.
ACC=Option	Option muß eine der folg. Möglichkeiten sein: IN-Datei muß schon Daten enthalten OUT-enthält die Datei Daten, so werden sie gelöscht NEW-Datei darf keine Daten enthalten UPD-Zeiger wird auf den Anfang gesetzt

Wenn keine Return-Variable vorhanden ist, werden Fehler, die während der Ausführung auftreten, in der normalen Weise behandelt. Es erscheint dann eine Fehlermeldung auf dem Dialoggerät. Wenn die Variable vorhanden ist, werden keine Fehler erzeugt und die Variable wird auf einen Wert gesetzt, der den Erfolg oder den Grund für einen Mißerfolg der FILE Anweisung anzeigt. Diese Werte werden in Tafel 10-2 aufgeführt.

Tafel 10-2

Return-Zustand bei Ausführung der FILE Anweisung

Return-Wert	Zustand
0	Alles in Ordnung
2	Datei existiert nicht
3	Dateiname war unzulässig
4	Unerlaubte Option aufgetreten
6	Nicht genug Speicher für Puffer vorhanden

Beispiele:

```
FILE #5; "XXX; ACC=IN; REC=8Ø",C
FILE #1; "DIDI"
```

Die Datei mit dem physischen Dateinamen XXX wird für wahlfreien Zugriff geöffnet, der Zeiger wird auf ~~das Ende~~ den Anfang der Datei gesetzt und C enthält den Return-Zustand nach Ausführung der Anweisung. Die Datei DIDI wird geöffnet, wobei der Zeiger auf den Anfang der Datei gesetzt wird.

10.2. Schließen von Dateien: CLOSE Anweisung

Alle Dateien werden automatisch bei Programm-Beendigung geschlossen. Eine Datei kann dynamisch während der Programm-Ausführung mit der CLOSE Anweisung geschlossen werden. Dies sollte dann getan werden, wenn es erforderlich ist, den Puffer-Speicherplatz für andere Dateien freizugeben.

Die CLOSE Anweisung löst die Zuordnung Dateiname-Dateinummer, die durch die FILE Anweisung hergestellt wurde, wieder auf und gibt alle Betriebsmittel, die zum Dateizugriff benötigt wurden, wieder frei.

Format

CLOSE #dateinummer,...,# dateinummer
CLOSE

Die angegebenen Dateinummern werden geschlossen.

Beschreibung

Wenn einer aufgeführten Dateinummer keine Datei zugeordnet war (durch vorhergegangene FILE Anweisung), wird keine Aktion ergriffen und auch keine Fehlermeldung gebracht.

Bei Verwendung der zweiten Form werden alle Dateien geschlossen.

10.3. Löschen von Dateien: ERASE Anweisung

Eine Datei kann durch eine ERASE Anweisung im System gelöscht werden.

Format

Das Format für die ERASE Anweisung ist:

ERASE dateiname [,return variable]

Der Dateiname ist ein Zeichenketten-Ausdruck. Die Return-Variable wird im Anschluß an die Ausführung der ERASE Anweisung ein Ergebnis enthalten.

Beschreibung

Die angegebene Datei wird gelöscht und kann nicht wiederverwendet werden.

Die Zahlwert-Variable in der Anweisung liefert das Ergebnis oder Zustand der ERASE Operation:

0	erfolgreiches Löschen
1	Datei wird zugegriffen und kann nicht gelöscht werden
3	eine solche Datei gibt es nicht

Beispiele:

```
10  ERASE "BFILE",N
20  PRINT N
 *  RUN
 0
```

Eine ERASE Anweisung wird zur Löschung der Datei BFILE benutzt. Das Ergebnis der Löschung wird ausgedruckt. Weil es eine erfolgreiche Löschung war, ist das Ergebnis 0.

```
10  DIM A$(96)
20  INPUT "GIB ZU LOESCHENDES PROGRAMM AN",A$
30  ERASE A$
```

Das obige Beispiel löscht ein BASIC Programm.

10.4. TRUNCATE Anweisung

Die TRUNCATE Anweisung bewirkt folgendes: Alle Bytes der Datei, die hinter der augenblicklichen Position des Cursors liegen, werden aus der Datei entfernt. Damit ist dieser Diskettenbereich für eine Wiederverwendung freigegeben.

Format

TRUNCATE #dateinummer

Beschreibung

Jedes Byte hinter der augenblicklichen Cursor-Position wird aus der Datei entfernt. Das Byte vor dem Cursor wird das letzte der Datei.

10.5. SPACE Anweisung

Die SPACE Anweisung wird zur Änderung der Cursor-Position innerhalb einer Datei benutzt. Der Cursor kann vorwärts oder rückwärts (in Richtung des Datei-Anfangs) um eine angegebene Zahl von Bytes oder bis zum Auftreten eines bestimmten Zeichens bewegt werden.

Format

SPACE #dateinummer;bewegungszähler
SPACE #dateinummer;bewegungszähler,suchzeichen
SPACE #dateinummer;bewegungszähler,suchzeichen,
return variable

Beschreibung

In der ersten Form wird der Cursor in der durch die angegebene Dateinummer bestimmten Datei um die durch den Bewegungszähler bestimmte Zahl von Bytes versetzt. Wenn der Bewegungszähler negativ ist, wird er in Richtung des Datenbeginns bewegt, wenn er gleich Null ist, hat die Anweisung keine Wirkung. Die Bewegung des Cursors wird durchgeführt, bis die angegebene Strecke zurückgelegt ist oder das Ende (bzw. der Anfang, falls der Zähler negativ ist) der Datei erreicht ist. Die EOF-Funktion zeigt an, ob die END-OF-FILE Marke erreicht wurde oder nicht.

In der zweiten Form läuft die Cursor-Bewegung in der oben beschriebenen Weise mit dem Zusatz ab, daß die Bewegung abbricht, wenn das Suchzeichen in der Datei erkannt wurde. Suchzeichen muß eine Zeichenkette der Länge 1 sein. Die

Cursor-Bewegung hört auf, wenn die angegebene Zahl von Bytes zurückgelegt ist, egal ob das Suchzeichen gefunden wurde oder nicht.

Die dritte Form arbeitet wie die zweite. Der Return-Variablen, die einfach oder indiziert ist, wird die Zahl der Bytes zugewiesen, die der Cursor tatsächlich zurückgelegt hat.

Beispiele:

```
  10  DIM A$(17)
  20  INPUT "DATEI:",A$
  30  FILE #1;A$ +"; ACC=IN"
  40  C%=0
  50  SPACE #1;32767," <13>"
  60  IF ~EOF(1) THEN DO
  70      C%=C%+1
  80      GOTO 50
  90  DOEND
 100  PRINT "<13> <10>ZAHL DER ZEILEN IN ";A$ ;" IST ";C%
*ASA-COUNT
*RUN
 DATEI: COUNT
 ZAHL DER ZEILEN IN COUNT IST 10
```

SPACE wird in obigem Beispiel zum Zählen der Zeilen der Datei COUNT benutzt.

```
  10  FILE #1;"TESTFILE"
  20  DIM A$(30)
  30  PRINT #1;"0123456789ABCDEF"
  40  RESTORE #1   \SETZE ZEIGER AN DEN ANFANG
  50  LINPUT #1;A$ \LESE UND DRUCKE EINE ZEILE
  60  PRINT A$
  70  RESTORE #1   \SETZE ZEIGER AN DEN ANFANG
  80  SPACE #1;5   \SETZE ZEIGER 5 BYTES VORWAERTS
  90  LET A$=" "
 100  READ #1;A$(1,6) \LESE UND DRUCKE DIE NAECHSTEN 6 By-
                       TES
```

```
110  PRINT A$
120  SPACE #1;50,"D",C \SETZE ZEIGER HINTER DAS NAECHSTE
                       "D"
130  LINPUT #1;A$
140  PRINT A$,C    \DRUCKE ZEICHENKETTE UND RETURN-
                    VARIABLE
150  SPACE#1;-10   \SETZE ZEIGER 10 BYTES (AB ENDE)
                    ZURUECK
155  REM MERKE:  CR GILT ALS ZEICHEN
160  LINPUT #1;A$
170  PRINT A$
180  CLOSE#1
190  ERASE "TESTFILE"
*RUN
0123456789ABCDEF
56789A
EF            3
789ABCDEF
```

10.6. Datei-Zugriff

In SIOS-BASIC gibt es 2 Dateizugriffsarten:
Sequentiellen und wahlfreien Zugriff. Bei sequentiellem Zugriff folgen die gelesenen oder geschriebenen Daten unmittelbar auf diejenigen des letzten Zugriffs. Jeder offenen Datei ist ein Zeiger (Cursor) zugeordnet, der auf das nächste Datenelement in der Datei zeigt, auf das zugegriffen wird.

Bei wahlfreiem Zugriff wird der jeweilige Satz angegeben, bei dem der Zugriff anfängt. In diesem Fall wird der Zeiger auf den Anfang dieses Satzes gesetzt.

Im SIOS-BASIC System können wahlfreier und sequentieller Zugriff für ein und dieselbe Datei kombiniert ausgeführt werden. Es ist z.B. möglich, den Zeiger auf den Anfang eines bestimmten, in der Anweisung angegebenen Satzes zu setzen und von diesem Punkt an sequentiell zuzugreifen.
Die Tatsache, daß wahlfrei zugegriffen werden soll, muß in

der FILE Anweisung für die Datei angegeben sein.

10.6.1. Sequentielles READ/INPUT/LINPUT auf Dateien

Die Anweisungen READ, INPUT und LINPUT für sequentiellen Zugriff lesen Daten von Dateien in Zahlwert- oder Zeichenketten-Variablen. Das erste Element, das gelesen wird, ist das, welches auf die augenblickliche Position des Zeigers folgt, d.h., direkt im Anschluß an den letzten Zugriff. Wie bei sequentiellem PRINT (Abschnitt 10.6.2.) werden Satzgrenzen ignoriert und die Liste der einzulesenden Datenelemente kann mitten in einem Satz anfangen und mitten in einem anderen aufhören.

Format

Die Formate für sequentiellen Datei-Lese-Zugriff sind:

INPUT # dateinummer;lese-element-liste
LINPUT # dateinummer;zeichenketten-variable
READ # dateinummer;lese-element-liste

Die Lese-Element-Liste ist eine Serie von durch Komma getrennten Variablen. Die gültigen Regeln für diese Liste sind dieselben, wie für die READ Anweisung in Abschnitt 5.9. beschrieben.

Beschreibung

Die Anweisung für sequentielle Datei-Eingabe liest KOI-7-Bit-Code Daten von einer Datei in derselben Weise wie durch die INPUT Anweisung-Daten über die Tastatur eingegeben werden.
Jedes Datenelement aus der angegebenen Datei wird in eine Variable in der Lese-Element-Liste gelesen, das erste Element in die erste Variable, das zweite in die zweite, und so weiter.

Das Ziel eines Zeichenketten-Wertes muß eine Zeichenketten-Variable sein; das Ziel eines Zahlenwertes muß eine Zahlenwert-Variable sein. Andernfalls tritt ein Fehler auf.
Wenn der einzulesende Zahlenwert nicht denselben Typ hat

wie die Variable, wird Typumwandlung durchgeführt, wie in Abschnitt 6.5., Tafel 6-2 beschrieben.

LINPUT # liest in eine Zeichenketten-Variable bis zu einem Wagenrücklauf-Zeichen (CR). Mit INPUT # Anweisung einzulesende Datenelemente müssen durch Komma getrennt sein.

Die READ Anweisung für sequentiellen Datei-Zugriff bringt binäre Daten von der angegebenen Datei in Variablen in der Lese-Element-Liste. Die Zahl der übertragenen Bytes ist genau die, die zur Füllung der Variablen benötigt wird. Es wird keinerlei Umwandlung oder Typ-Kontrolle durchgeführt. Die Zahl der jeweils übertragenen Bytes wird in Tafel 10-3 erläutert.

Wenn eine EOF-Bedingung auftritt, bleiben die Variablen unverändert und die EOF Funktion (Abschnitt 10.7.) bekommt den Wert "wahr".

Lesen von Zeichenketten

Wenn eine Zeichenkette von einer binären Datei gelesen wird, hängt die Zahl der gelesenen Zeichen von der Form der Variablen ab. Wenn z.B. A$ eine einfache Zeichenketten-Variable ist, dann:

READ #1;A$	liest die physische Länge von A$
READ #1;A$ (I)	liest die physische Länge der Teilkette, die bei I anfängt
READ #1;A$ (I,J)	liest J-I+1 Zeichen in die Teilkette, die bei I anfängt.

Bei Eingabe von Zeichenketten mit INPUT werden überflüssige Zeichen überlesen, falls die Zeichenketten-Variable nicht lang genug ist, das ganze Datenelement aufzunehmen.

TAFEL 10-3

Datentyp	Zahl der Bytes auf der Datei
INTEGER	2
REAL	8
ZEICHENKETTE	aktuelle Zahl der gelieferten Zeichen: logische (physische Größe bei READ) Größe der Zeichenkette wenn keine Teilkette oder angegebene Größe der Teilkette

10.6.2. Sequentielles PRINT und WRITE

Die sequentiellen PRINT und WRITE Anweisungen für Dateien schreiben Daten-Elemente auf eine Datei, beginnend bei der augenblicklichen Position des Zeigers. Die Datenelemente können Zahlwert- oder Zeichenketten-Ausdrücke sein.

Format

Die Formen von sequentiellen Ausgabe-Anweisungen auf Dateien sind:

```
PRINT  #dateinummer;ausgabe-liste
PRINT  #dateinummer
WRITE  #dateinummer;ausgabe-liste
```

Die Ausgabe-Liste ist eine Folge von Zahlwert- und/oder Zeichenketten-Ausdrücken. Die Regeln für den Aufbau einer solchen Liste sind dieselben wie für die PRINT Anweisung in Abschnitt 5.8. beschrieben.

Wenn die Ausgabe-Liste bei der PRINT Anweisung weggelassen wird, wird ein RETURN auf die Datei geschrieben (Zeilenvorschub wie bei einer PRINT Anweisung).

Beschreibung

Jedes Element in der Ausgabe-Liste wird auf die Datei geschrieben. Dies geschieht in der Reihenfolge, die durch die Anweisung vorgegeben ist. Die Elemente werden ab der Position geschrieben, auf der der Zeiger sich gerade be-

findet, wobei alle Daten, die sich zuvor dort befinden, überschrieben werden. Satzgrenzen werden ignoriert; eine sequentielle Ausgabe kann mitten in einem Satz beginnen und mitten in einem anderen Satz aufhören.

Die durch PRINT geschriebenen Daten sind genau die KOI-7-Bit-Zeichen, die bei Ausführung eines gleich aufgebauten PRINT zur Ausgabe über Bildschirm oder Drucker kommen. Man merke sich, daß man mit einer INPUT Anweisung nicht diejenigen Daten zurücklesen kann, die zuvor mit einer PRINT Anweisung auf eine Datei geschrieben wurden, wenn nicht Kommas zwischen die Elemente gesetzt und Zeichenketten mit Anführungszeichen geklammert wurden.

Eine sequentielle WRITE Anweisung bringt binäre Daten aus Elementen der Ausgabeliste auf die angegebene Datei. Der Umfang (in Bytes) der übertragenen Daten hängt von der Größe der Datenelemente (siehe Tafel 10-3) ab. Es wird keine Typumwandlung durchgeführt und es ist nicht möglich, Struktur oder Typ der Daten auf der Datei aus der Information allein zu ermitteln. So geschriebene Daten werden mit der READ Anweisung wieder eingelesen.

10.6.3. RESTORE Anweisungen für Dateien

Die Restore Anweisung für Dateien positioniert den Dateizeiger auf den Anfang der Datei. Diese Anweisung kann für jede Datei verwendet werden. Die RESTORE Anweisung für Dateien mit wahlfreiem Zugriff positioniert den Dateizeiger auf den Anfang jedes beliebigen Satzes.

Format

```
RESTORE #dateinummer
RESTORE #dateinummer,satznummer
```

Die Dateinummer bezeichnet eine Datei, die augenblicklich offen ist.
In der zweiten Form gibt die Satznummer den Satz an, auf dessen Anfang der Datenzeiger gesetzt wird.

Beschreibung

Wenn RESTORE für eine Datei ausgeführt wird, wird der Dateizeiger auf den Anfang des ersten Satzes der Datei gesetzt.

Wenn RESTORE für eine Datei mit wahlfreiem Zugriff ausgeführt wird, wird der Dateizeiger auf den Anfang des angegebenen Satzes gesetzt.

Beispiel:

```
  5  FILE #1;"AFILE"
 10  FILE #2;"BFILE"
 20  PRINT #1;123.4
 30  WRITE #2;567.8
 40  RESTORE #2
 50  RESTORE #1
 60  INPUT #1;C
 70  READ #2;D
 80  PRINT C,D
*RUN
 123.4            567.8
```

Wenn die RESTORE Anweisungen ausgeführt werden, wird der Zeiger von Datei 2 auf den Anfang der Datei zurückgesetzt. Dann wird der Zeiger von Datei 1 auf den Anfang der Datei gesetzt. Auf Datei 1 wird als KOI-7-Bit-Code Datei zugegriffen, wobei INPUT und PRINT Anweisungen benutzt werden. Auf Datei 2 wird als binäre Datei mit READ und WRITE Anweisungen zugegriffen.

10.6.4. READ/INPUT/LINPUT auf wahlfreie Dateien

Die Anweisungen READ, INPUT und LINPUT für Dateien mit wahlfreiem Zugriff lesen Datenwerte ab einem bestimmten Satz einer bestimmten Datei und weisen diese Werte Variablen zu.

Format

Die Formen von READ, INPUT und LINPUT Anweisungen sind:

```
READ   # dateinummer,satznummer;eingabeliste
INPUT  # dateinummer,satznummer;eingabeliste
LINPUT # dateinummer,satznummer;zeichenkettenvariable
```

Die Dateinummer und Satznummer sind ganzzahlige Ausdrücke. Die Eingabeliste hat dieselbe Form wie in einer READ Anweisung (Abschnitt 5.9.).

Beschreibung

Datenwerte werden von dem angegebenen Satz gelesen und den Variablen in der Eingabeliste zugewiesen. Wenn eine Satznummer in der Datei nicht vorkommt (zu große Nummer, die die Zahl der Sätze übersteigt), erscheint die EOF-Bedingung.

Zum Setzen des Dateizeigers auf den Anfang eines bestimmten Satzes, ohne daß Daten eingelesen werden, wird die RESTORE Anweisung für Dateien mit wahlfreiem Zugriff verwendet (Abschnitt 10.6.3.).

10.6.5. PRINT und WRITE auf wahlfreie Dateien

Die Anweisungen PRINT und WRITE für Dateien mit wahlfreiem Zugriff schreiben eine Liste von Datenelementen auf eine angegebene Datei. Die Ausgabe beginnt bei einem in der Anweisung angegebenen Satz. Daten, die vor oder hinter dem angegebenen Gebiet liegen, werden nicht verändert.

Format

Die Formen der PRINT und WRITE Anweisungen für wahlfreie Dateien sind:

```
PRINT  #dateinummer,satznummer;ausgabeliste
WRITE  # dateinummer,satznummer;ausgabeliste
```

Sowohl dateinummer als auch satznummer sind ganzzahlige Ausdrücke. Die Ausgabeliste hat dasselbe Format wie bei PRINT auf sequentiellen Dateien. Hier kann sie jedoch nicht entfallen.

Beschreibung

Die Anweisungen PRINT und WRITE für wahlfreie Dateien positionieren den Zeiger auf den Anfang des angegebenen Satzes und schreiben dann den Inhalt der Ausgabeliste.

Der erste Satz der Datei hat die Nummer Ø.

PRINT Anweisungen für sequentielle und wahlfreie Dateien können benutzt werden, um auf dieselbe Datei zu schreiben. Dasselbe gilt für WRITE. Ein sequentielles PRINT, das auf ein wahlfreies PRINT folgt, wird seine Daten in unmittelbarem Anschluß an die vorhergegangenen Daten setzen.

10.7. EOF Funktion

EOF (x)

EOF zeigt an, ob für die Datei mit der Nummer x die end-of-file- (Dateiende-) Bedingung vorliegt. Die Funktion bringt 1, falls die Bedingung vorliegt, 0 falls nicht. Die Dateinummer liegt zwischen 1 und 15.

Beispiel:

```
  5  DIM T$(62)
 10  FILE #1;"DATEI".R%
 20  PRINT R%
 25  INPUT " ",T$
 26  IF T$="" THEN  50
 30  PRINT #1;T$
 35  GOTO  25
 50  RESTORE #1
 55  PRINT
 60  INPUT #1;T$
 62  IF EOF(1) THEN  100
 65  PRINT T$
 70  GOTO  60
100  CLOSE #1
110  ERASE "DATEI",R%
120  PRINT R%
```

11. Formatierte Ausgabe

Die PRINT USING Anweisung in SIOS BASIC gibt dem Benutzer explizite und genaue Kontrolle über das Format der Programm-Ausgabe. Jeder Typ von Zahl kann gedruckt werden: ganzzahlig und Gleitkomma. Die genaue Position des Vorzeichens kann angegeben werden. Zeichenketten-Werte können in bestimmte Felder gedruckt werden. Text und Leerzeichen können wo gewünscht eingefügt werden. Wagenrücklauf (CR) und Zeilenvorschub (LF) unterliegen expliziter Kontrolle.

Zur Beschreibung des Ausgabe-Formats werden Format-Zeichenketten benutzt. Diese Zeichenketten sind Bestandteil der PRINT USING Anweisung.

11.1. PRINT USING Anweisung

Die PRINT USING Anweisung erlaubt dem Benutzer, Daten formatiert auszugeben.

Format

Die Form von PRINT USING ist:

PRINT print-using-liste

Die Print-Using-Liste ist eine Liste von Ausdrücken und Funktionen, deren Werte gedruckt werden. Die Floskel "USING zeichenketten-ausdruck" kann irgendwo eingeschoben sein. Ansonsten ist die Print-Using-Liste wie die Liste in der PRINT Anweisung (siehe Abschnitt 5.8.) aufgebaut.

Die Zeichenkette hinter USING dient als Format-Beschreibung für die Ausgabe.

Beschreibung

Die Format-Beschreibung gibt die Form an, in der die Datenelemente der Print-Using-Liste ausgegeben werden. Die vollständige Beschreibung der Format-Zeichenkette ist in Abschnitt 11.2. enthalten.

Jedes Komma in der Print-Using-Liste ist nur Trennzeichen; es hat keine Format-beeinflussende Funktion wie in PRINT.

Bei der Ausführung einer USING-Formel wird jede Angabe in der zugeordneten Format-Beschreibung herausgezogen und untersucht. Wenn die Angabe einen Zahlenwert verlangt, wird die Print-Using-Liste auf den zugehörigen Ausdruck hin untersucht. Jeder Ausdruck wird gemäß der zugehörigen Spezifikation in der Format-Beschreibung ausgegeben. Jede Druck-Funktion in der Liste wird ausgeführt, wenn sie auftritt.

Wenn der Ausdruck und die Spezifikation nicht zueinander passen, wird das Programm mit einer Fehlermeldung abgebrochen. Wenn der Wert eines Zahlwert-Ausdrucks größer ist als mit dem Format ausgebbar, wird die Zahl ohne Formatierung ausgegeben; vorgestellt werden zwei Sternchen und die Ausgabe erfolgt auf der nächsten Zeile. Das Programm fährt fort. Bei Ausgabe einer Gleitkommazahl mittels einer INTEGER-Format-Spezifikation wird auf die nächste ganze Zahl gerundet.

Wenn das Ende der Format-Beschreibung erreicht wird, bevor die Print-Using-Liste erschöpft ist, geht es wie bei einer normalen PRINT Anweisung weiter, bis eine neue USING Formel ermittelt wird.

11.2. Format-Zeichenketten

Im folgenden werden die Format-Zeichen und ihre Funktion angegeben:

Zeichen	Ausgabe	Kommentar
#	Ziffer oder Leerzeichen oder "-"	Ausgabe eines Leerzeichens, sofern führende oder nachfolgende Stelle Null."-" Vorzeichen ergibt sich, wenn Zahl negativ.
D	Ziffer	Ausgabe einer Null, sofern führende oder nachfolgende Stelle Null.
+	"+" oder "-"	Vorzeichen
-	"-" oder Leerzeichen	Leerzeichen, wenn die Zahl nicht negativ ist.
¤	"¤" oder Ziffer	Ziffern an den Stellen mit dem Zeichen "¤" überschreiben dieses Zeichen. Es wird nur ein¤ vor der Ziffernfolge ungleich Null ausgegeben, an den anderen Stellen erscheint anstelle von¤das Leerzeichen.
*	"*" oder Ziffer	Ausgabe eines "*" in führenden oder nachfolgenden Stellen, sofern Wert Null, sonst Ausgabe der Ziffer.
P	"."	gibt die Position des Dezimalpunktes in einer Zahl an
.	"." oder Leerzeichen	Ausgabe eines ".", wenn es keine ganze Zahl ist, ansonsten Ausgabe Leerzeichen. Eine Zahl ist dann keine ganze Zahl, wenn Ziffern auf der rechten Seite nach dem Dezimalpunkt ausgegeben werden sollen.

Zeichen	Ausgabe	Kommentar
,	"," oder Leerzeichen	Ausgabe Leerzeichen, wenn links vor dem Komma keine Ziffern stehen, sonst Komma.
^^^^	Esdd	Für s steht "+" oder "-", dd sind Ziffern, veranlaßt die Zahl in Exponentenform auszugeben.
^^^^^	ESddd	Wie bei"^^^^", nur mit dem Unterschied, daß hier der Exponent aus 3 Ziffern besteht.

Anmerkungen:

1) "P" oder "." darf nur einmal vorkommen
2) "," darf nicht rechts von einem "P", "," oder "^" stehen
3) ^^^^und^^^^^ können nur auf der rechten Seite einer Format-Zeichenkette stehen
4) Wenn "+" oder "-" auftreten, wird das "#" niemals als Vorzeichen verwendet
5) Wenn kein "+" oder "-" auftritt, dann wird ein "#" als Vorzeichen in der Formatzeichenkette verwendet
6) Wenn kein "P" oder "." auftritt wird angenommen, daß der Dezimalpunkt rechts von der "#" oder "D" Zeichenkette stehen

Beispiele:

Format	Zahl	Ausgabe
##.##	123	123
	123.5	123.5
	123.526	123.53
	-12	-12
###.DD	123	123.00
	5.6	5.60
##DD.D	124	124.0
	4	04.0

Format	Zahl	Ausgabe
+ # # # #	3	|+ 3|
	1234	|+1234|
	-2	|- 2|
	23.6	|+ 24|
- # # # #	4	| 4|
	-71	|- 71|
	0	| 0|
## -	0	| 0|
	-4	| 4-|
	-23	|23-|
##.DD+	0	| 0.00+|
	-2.4	| 2.40-|
	12.34	|12.34+|
###P##	123	|123.|
¤###.DD	4.2.	|¤ 4.20|
	123.45	|¤123.45|
	.5	|¤ .50|
¤¤¤¤ DD	1.267	| ¤1.27|
	234	|¤234.00|
	9876.5	|9876.50|
¤¤¤¤.DD	4.2	|¤¤¤4.20|
	123.45	|¤123.45|
	.5	|¤¤¤¤.50|
##,###,###.D	1234	| 1,234.0|
	1000000	|1,000,000.0|
#.### ^ ^ ^ ^	1	|1 E+00|
	234.5	|2.345E+02|
##.DD ^ ^ ^ ^ ^	0	| .00E+000|
	1	|10.00E-001|
	3456	|34.56E+002|
	-5E+123	|-5.00E+123|

Format	Zahl	Ausgabe
####,,DDD.DD¤		
	1	001.00¤
	12345	12,,345.00¤
	.003	000.003
DD##,##+#--++#		
	1	00 , + ++1
	-34	00 , -3----4

12. Überlagerung von Programmen

Weil die maximale Größe eines SIOS BASIC Programms notwendigerweise durch den verfügbaren Speicher begrenzt ist, besitzt SIOS BASIC Sprachmittel zur Zerlegung von Programmen in Einheiten, die sich gegenseitig aufrufen können. Jede Einheit muß auf die Diskette ausgelagert werden; von dort kann sie durch das aktuell laufende Programm in dem Benutzer-Arbeitsbereich geladen und aufgerufen werden.

Zum Aufruf eines anderen Programmes wird die CHAIN Anweisung benutzt. Die COM Anweisung ermöglicht die gemeinsame Verwendung von Variablen durch mehrere Programme.

12.1. CHAIN Anweisung

Die CHAIN Anweisung beendet das aktuelle Programm und beginnt die Ausführung eines anderen Programms.

Format

CHAIN zeichenketten-ausdruck

Der ausgewertete Zeichenketten-Ausdruck ergibt den Namen eines SIOS BASIC Programms, das sich auf der Diskette des Benutzers befindet. Die Ausführung beginnt bei der ersten ausführbaren Anweisung im gerufenen Programm.

Beschreibung

CHAIN ruft das Programm, das durch den Zeichenketten-Ausdruck bezeichnet wird und es ersetzt das aktuelle Programm. Wenn das durch CHAIN gerufene Programm anhält, wird nicht automatisch zu dem aufrufenden Programm zurückgekehrt. Das aufgerufene Programm kann seinerseits ein anderes Programm mittels CHAIN aufrufen, einschließlich derjenigen, von dem es selbst aufgerufen wurde.

Nur in COM Anweisungen deklarierte Variablen werden während der CHAIN Operation gerettet. Alle Variablen und Felder des aktuellen Programms, die nicht in COM deklariert wurden, gehen verloren, wenn das neue Programm seine Ausführung beginnt.

Alle Dateien, die im aktuellen Programm geöffnet wurden, bleiben offen.

Beispiele:

```
*10  PRINT "HALLO PIT"
*20  CHAIN "PIT"
*ASAVE-PAT
*NEW
*10 PRINT "HALLO PAT"
*ASAVE-PIT
*NEW
*XEQ-PAT
HALLO PIT

HALLO PAT
```

Das Hauptprogramm PAT ruft das Programm PIT mit der CHAIN Anweisung in Zeile 20 auf. Danach beginnt die Ausführung von PIT; sie endet bei der letzten Zeile von PIT.
Es wird kein Variablenwert aus PAT gerettet, nachdem CHAIN "PIT" ausgeführt wurde.

12.2. COM Anweisung

Die COM-Anweisung wird benutzt, um Daten-Werte zwischen segmentierten Programmen zu übergeben. Die in einer COM-Anweisung aufgeführten Variablen werden in einem gemeinsamen Speicherbereich abgelegt, so daß auf Werte, die diesen Variablen in einem Programm zugewiesen wurden, in anderen Programmen zugegriffen werden kann (sofern Programmwechsel mittels der CHAIN-Anweisung erfolgte).
COM Anweisungen müssen in einem Programm vor jeder anderen Anweisung stehen (Ausnahme: REM Anweisungen). Jede Dimensionierung von Variablen wird innerhalb der COM Anweisung vorgenommen. Variable, die in einer COM Anweisung auftreten, dürfen nicht in einer DIM Anweisung in demselben Programm vorkommen.

Format

Das Format der COM Anweisung ist

COM liste

Die Liste besteht aus Variablen-Deklarationen. Einfache Variablen werden durch den Variablennamen, Felder durch den Feldnamen und die Feldgrenzen angegeben. Die Feldgrenzen werden wie in einer DIM Anweisung angegeben.

Der Typ der in der Liste deklarierten Variablen wird mit REAL angenommen, es ei denn, der Variablenname endet auf "¤" (bei Zeichenketten-Variablen) oder "%" (bei INTEGER-Variablen).
Felder und einfache Variablen, die mit COM deklariert sind, werden mit Ø vorbesetzt. Zeichenketten, die mit COM deklariert sind, werden als leere Zeichenkette vorbesetzt. Solche gemeinsame Variablen dürfen nicht zusätzlich mit DIM deklariert werden.
COM-Listen brauchen NICHT in verschiedenen Programmen die gleiche ORDNUNG der Variablennamen zu besitzen. Aber Variablen, die von verschiedenen Programmen aus benutzt werden sollen, müssen gleich sein in Namen und Dimensionierung.

13. Im Tischrechnermodus ausführbare Anweisungen

Im allgemeinen muß jeder Anweisung eine Anweisungsnummer vorausgehen und sie kann nur im Rahmen eines Programms ausgeführt werden. Einige Anweisungen jedoch können direkt von der Tastatur aus ausgeführt werden. Dieser Modus - Tischrechnermodus genannt - kann zum Zwecke der Fehlersuche oder einfacher Berechnungen (z.B. Ausgabe der Werte von Variablen und Ausdrücken) nützlich sein. Die folgenden Anweisungen können im Tischrechnermodus ausgeführt werden:

DIM
REM
PRINT
READ
WRITE
RESTORE
FILE
RANDOMIZE
LET
ERASE
SYSTEM

Beschreibung

Wenn eine der obigen Anweisungen ohne voranstehende Anweisungsnummer eingegeben wird, wird sie unmittelbar ausgeführt. Wenn die Anweisung eine PRINT Anweisung ist, erfolgt die Ausgabe über das Dialoggerät.
Wenn eine Programmausführung durch Betätigung der DEL-Taste oder durch STOP Anweisung unterbrochen wurde, kann durch Ausdrucken von Variablen festgestellt werden, in welchem Zustand das Programm sich befand, als es abgebrochen wurde. Wird das Programm in einer Mehrzeilen-Funktion angehalten, so haben die formalen Parameter-Variablen, die innerhalb der Funktion verwendet werden, die Werte, wie sie in der Funktion verwandt wurden. Dies gilt selbst dann, wenn es globale Variablen mit gleichem Namen gibt.

```
Beispiel: *10  LET A=1
          *20  STOP
          *30  PRINT A
          RUN
          STOP AT 20
          PRINT A
           1
          *LET A=4
          *CONTINUE
          4
          READY
          *
```

14. Aufruf von Assemblerunterprogrammen

Mittels der CALL-Anweisung ist es möglich, Unterprogramme aufzurufen, die in der Assemblersprache des U880 oder in Makrobefehlssprache MABS 1520 geschrieben sind.
Für letztere ist zu beachten, daß nur die Makrobefehle abgearbeitet werden können, deren Makrointerpreter-Bestandteil generiert sind.

Das Format der CALL-Anweisung ist:

```
CALL up-name [,übergabeparameterliste]
             [;rückgabeparameterliste]
```

Der UP-Name ist ein Zeichenkettewert. Die Elemente der Parameterliste sind durch Komma zu trennen.
Übergabeparameter sind Ausdrucke vom Zahl- oder Zeichenkettentyp. Rückgabeparameter sind Variable, es sind ebenfalls alle 3 Typen zugelassen.

```
Beispiel: 1Ø CALL "TEST"
          5Ø CALL "PROG", I%, J%, Ko; A1, Xo
```

Im Assemblerunterprogramm-Rahmen sind die Namen der UP aufzuführen, die für eine Abarbeitung gleichzeitig zur Verfügung stehen sollen.
Zu jedem Unterprogramm sind die Parameter hinsichtlich Anzahl und Typ zu beschreiben, gleichzeitig erfolgt die Angabe der Anfangsadresse des Assemblerunterprogramms.
Zum Zeitpunkt der Abarbeitung können die von BASIC übergebenen Werte über den Stack erreicht werden. Dabei wird mittels eines Zeigers auf eine Tabelle der Anfangsadressen der Werte verwiesen.
Die Rückgabe der Werte erfolgt dadurch, daß im Stack der Zeiger auf eine Tabelle der Anfangsadressen der Rüchgabewerte eingestellt sein muß.

Anhang B

SIOS-BASIC-Anweisungen (Zusammenfassung)

CHAIN - Beendet laufendes Programm und arbeitet das im Statement angegebene Programm ab. Variable werden über COM-Statement übergeben.

CLOSE - Schließen aller Files

COM - Deklaration von globalen Variablen (Parameterübergabe)

DATA - Bereitstellen von Daten

DEF - Funktions-Definition

DIM - Speicherplatzreservierung von Feldern

DO... DOEND- Zusammenfassen von Statements nach IF...THEN oder ELSE

ELSE - In Verbindung mit IF...THEN gebraucht

END - Beendet die Ausführung eines laufenden Programms

ERASE - Löschen eines spezifizierten Files

FILE - Zuweisung eines File-Namens zu einer File-Nummer

FNEND - Beendet eine Mehrzeilenfunktionsdefinition

FOR... NEXT- Wiederholung einer Gruppe von Anweisungen

GOTO - Verzweigung zur spezifizierten Anweisungs-Nummer

GOSUB - Aufruf eines Unterprogramms

IF... THEN - Berechnung des Ausdrucks und Verzweigung, sofern Bedingung erfüllt

INPUT - Eingabe von Zeichenketten oder numerischen Werten über die Tastatur

INPUT # - Eingabe von Zeichenketten oder numerischen Werten von einer Datei

LET - Zuweisung von Werten an Variable oder Feldelemente

LINPUT - Eingabe von Zeichenketten an Zeichenkettenvariable

LINPUT # - Zuweisung von Daten aus einem File an Zeichenkettenvariable

NEXT - Beendigung einer Schleifenanweisung

ON... GOSUB - Mehrfachverzweigung zu Unterprogrammen in Abhängigkeit vom Wert eines Ausdruckes

ON... GOTO - Mehrfachverzweigung zu Anweisungen in Abhängigkeit vom Wert eines Ausdruckes

ON...RESTORE- Einstellen des Zeigers im Data-Statement

PRINT - Drucken des Inhalts einer Liste von numerischen oder Zeichenketten-Ausdrücken

PRINT # - Ausgabe des Inhalts einer Liste von numerischen oder Zeichenketten-Variablen in die spezifische Datei

PRINT USING - Drucken des Inhalts einer Liste von Variablenwerten mit Format-Kontrolle

RANDOMIZE - Ändern der Ausgangszahl für die Funktion RND

READ - Zuweisung von Werten an Variable ausgehend vom DATA-Statement

READ # - Lesen von einem oder mehreren Elementen eines Binärfiles und Wertzuweisung an Variable

REM - Kommentaranweisung

RESTORE - Rücksetzen des Datenzeigers

RESTORE # - Rückpositionierung des Dateizeigers

RETURN - Rückgabe der Kontrolle von Unterprogrammen

SPACE - Bewegung des Kursors in einer Datei

STOP - Ende der Ausführung

SYSTEM - Steuerung systemabhängiger Funktionen

TRUNCATE - Setzen EOF einer Datei
WRITE # - Ausgabe des unkonvertierten binären Inhalts in eine Datei

Anhang C

SIOS-BASIC-Kommandos (Zusammenfassung)

APPEND	- Anfügen eines Programmteiles (mittels ASA auf Diskette gespeichert)
ASAVE	- Speichern eines Programms auf FD
CLEAR	- Löschen von Variablen, Schließen von Dateien
CONTINUE	- Fortsetzung eines Programms nach STOP
DELETE	- Löschen von Programmzeilen
GET	- Laden eines BASIC-Programms von FD
LIST	- Auflisten eines Programms
NEW	- Löschen des Programm-Speichers
QUIT	- Ende einer BASIC-Sitzung
RENUMBER	- Neue Numerierung aller Quellzeilen
RUN	- Ausführung eines Programms
SAV	- Speichern eines Programms in compilierter Form
STEP	- Schrittweise Abarbeitung eines Programms
SIZE	- Statusinformationen
XEQ	- Holen und Abarbeiten eines Programms (GET, RUN)

Anhang D

SIOS-BASIC-Funktionen

ABS(x)	- absoluter Wert von x
ASC(s)	- Code-Zeichen für das erste Zeichen eines Ausdrucks
ATN(x)	- Arcustangens von x
CHR$(x)	- Generierung eines Zeichens
COS(x)	- Cosinus von x
EOF(x)	- EOF-Bedingung einer Datei
EXP(x)	- e^x
INT(x)	- größte Zahl kleiner oder gleich x
LEFT$(s,n)	- Teilzeichenkette
LEN(s)	- logische Länge einer Zeichenkette
LOG(x)	- natürlicher Logarithmus von x
POS(s1,s2)	- Teilzeichenreihe suchen
RIGHT$(s,n)	- Teilzeichenkette
RND	- Pseudozufallszahlen-Generierung
SEG$(s,n,m)	- Teilzeichenreihe
SGN(x)	- Vorzeichen-Ermittlung
SIN(x)	- Sinus
STR$(x)	- Konvertierung Wert in Zeichenkette
SQR(x)	- Quadratwurzel
TAB(x)	- Positionierung
TAN(x)	- Tangens
VAL(s)	- Konvertierung Zeichenkette in Zahlenwert

Anhang E

Liste der Fehlernummern und Erläuterung

Gruppe 1: Übersetzungszeitfehler

Fehlernummer	Erläuterung
1	Fehlerhafte Anweisungsnummer
2	Eingabe nicht erkennbar
3	Zahl der öffnenden und schließenden Klammern verschieden
4	Literal zu lang
5	Unzulässige Anweisung in zweiter Klausel
6	Ausdruck zu komplex
7	Fehlerhaftes Element in Ausdruck
8	Fehlendes schließendes Ausführungszeichen
9	Falscher Anwenderfunktionsname
10	Zeichen nach Anweisungsende
11	Fehlendes "#"
12	Fehler im Ausdruck für die Dateibeschreibung
13	Fehlendes ";"
14	Fehlender oder falscher Dateiname
15	Unzulässige Return-Variable
16	Unzulässiger Ausdruck für die Datensatznummer
17	Fehlende oder unzulässige Anweisungsnummer
18	Unzulässiger Auswahlausdruck
19	Unzulässiger Funktionsname
20	Unzulässige THEN, ELSE Klausel
21	Unzulässiges Zuweisungsobjekt
22	Fehlender Zuweisungsoperator
23	Unzulässiger Ausdruck
24	Unzulässige Variable
25	Unzulässiges Listenelement
26	Unzulässiger formaler Parameter

Fehlernummer	Erläuterung
27	Als FOR/NEXT Index sind keine einfachen Variablen angegeben
28	Unzulässige "USING" Zeichenkette
29	LINPUT Variable muß vom Typ Zeichenkette sein
30	Fehlendes "-"
31	Fehlender oder unzulässiger Anfangswert
32	Fehlendes "TO"
33	Fehlender oder unzulässiger Endwert
34	Fehlende oder unzulässige Schrittweite
35	Zergliederung fehlt
36	Fehlendes "THEN"
37	Unzulässige Funktionsdefinition
38	Kann im Tischrechner nicht ausgeführt werden
39	Unzulässiges Objekt
40	Kommando nicht erlaubt in Verbindung mit Datei
41	Kompilierte Datei in unzulässigem Zusammenhang
42	Undefinierte Umgebung, keine Fortsetzung möglich
43	Keine BASIC Datei
44	Unzulässiger Parameter

Gruppe 2: Programmstrukturfehler

Fehlernummer	Erläuterung
50	Bezugnahme auf undefinierte Variable oder undimensioniertes Feld
51	Bezugnahme auf nicht existierende Zeilennummer
52	Bezugnahme auf undefinierte Funktion
53	Bezugnahme auf nichtdeklarierte Dateinummer
54	Geschachtelte Funktionsvereinbarungen sind unzulässig

Fehlernummer	Erläuterung
55	Unzulässige Zahl von Puffer
56	Unzulässige Dateinummer
57	Unpaarige Zahl DO/DOEND's
58	RETURN ohne frühers GOSUB
59	FNEND ohne RETURN
60	NEXT ohne FOR
61	Unzulässige Verschachtelung von FOR/NEXT
62	NEXT nicht im selben Block mit FOR
63	INPUT/READ kann keine Funktionen einlesen
64	Fehlerhafter Anwenderfunktionsaufruf
65	Typ unpassend
66	Dimension zu groß
67	Zeichenkette darf nicht rückdimensioniert werden
68	Unkorrekte Zahl von Argumenten bzw. Indizes
69	Unkorrekte Zahl von Parameter
70	Bezugnahme auf eine undefinierte Prozedur
71	Unzulässige Begrenzungskette
72	Unzulässiger TAB Gebrauch

Gruppe 3: Systembegrenzungen und -störungen

Fehlernummer	Erläuterung
80	Symboltabelle gefüllt
81	Zu viele Dateien eröffnet
82	Speicherüberlauf
83	Runtime-Keller Überlauf
84	DO Verschachtelung zu tief
85	Ungenügende Betriebsmittel-Bereitstellung
86	Merkmal nicht implementiert
87	Interpreter-Fehler
88	Interface-Fehler

Gruppe 4: Fehler in Verbindung mit Feldern und Zeichenketten

Fehlernummer	Erläuterung
100	Argument außerhalb eines Wertbereiches
101	Unzulässige Teilzeichenreihenbeschreibung
102	Indexvariable außerhalb eines Wertbereichs
103	Zweite Indexvariable außerhalb eines Wertebereichs
104	Versuch die Dimension zu vergrößern
105	Fehlende Indizes

Gruppe 5: Ein-/Ausgabe-Fehler

Fehlernummer	Erläuterung
120	Datei existiert nicht
121	Datei existiert
122	Positionierung vor Beginn der Datei
123	Positionierung über Ende der Datei hinaus
124	Außerhalb von DATA
125	Unzulässiger Dateiname
126	Unzulässiger Dateityp
127	Dateischutzfehler
128	Datei bereits eröffnet
129	Nichtzugewiesene Ein-/Ausgabe
130	Datei nicht eröffnet
131	Dateifehler
132	Diskettenfehler
133	Diskette nicht bereit
134	Diskette gefüllt
135	Ungültige Operation
136	Numerischer Umwandlungsfehler
137	Ungenügende Eingabe
138	Dateistruktur falsch

Gruppe 6: Warnungen

Fehlernummer	Erläuterung
140	Unzulässige Zahl
141	Überlauf
142	Warnung:Zahl zu klein
143	Division durch Null
144	Quadratwurzel von einer negativen Zahl
145	Logarithmus von einer negativen Zahl oder Null
146	Zeichenkette wird während Zuweisung abgeschnitten
147	Format zu klein um Zahl aufzunehmen
160	Unzulässiges Formatzeichen
161	Unzulässiges Exponentenfeld
162	keine Ziffernpositionen
163	keine Formatzeichenkette

Anhang A

KOI-7-Bit-Code

Dezimalwert	Darstellung	Erläuterung für nicht darstellbare Zeichen
0		NUL
1		SOH
2		STX
3		ETX
4		EOT
5		ENQ
6		ACK
7		BEL
8		BS
9		HT
10		LF
11		VT
12		FF
13		CR
14		SO
15		SI
16		DLE
17		DC1
18		DC2
19		DC3
20		DC4
21		NAK
22		SYN
23		ETB
24		CAN
25		EM
26		SUB
27		ESC
28		FS
29		GS
30		RS
31		US

Dezimalwert	Darstellung	Erläuterung für nicht darstellbare Zeichen
32		Leerzeichen
33	!	
34	"	
35	#	
36	¤	
37	%	
38	&	
39	'	
40	(	
41	)	
42	*	
43	+	
44	,	
45	-	
46	.	
47	/	
48	0	
49	1	
50	2	
51	3	
52	4	
53	5	
54	6	
55	7	
56	8	
57	9	
58	:	
59	;	
60	<	
61	=	
62	>	
63	?	
64	@	
65	A	

Dezimalwert	Darstellung	Erläuterung für nicht darstellbare Zeichen
66	B	
67	C	
68	D	
69	E	
70	F	
71	G	
72	H	
73	I	
74	J	
75	K	
76	L	
77	M	
78	N	
79	O	
80	P	
81	Q	
82	R	
83	S	
84	T	
85	U	
86	V	
87	W	
88	X	
89	Y	
90	Z	
91	[	
92	\	
93	]	
94	^	
95	_	
96	`	
97	a	
98	b	
99	c	

Dezimalwert	Darstellung	Erläuterung für nicht darstellbare Zeichen
100	d	
101	e	
102	f	
103	g	
104	h	
105	i	
106	j	
107	k	
108	l	
109	m	
110	n	
111	o	
112	p	
113	q	
114	r	
115	s	
116	t	
117	u	
118	v	
119	w	
120	x	
121	y	
122	z	
123	{	
124	|	
125	}	
126	~	
127		DEL

NOTIZEN

NOTIZEN

NOTIZEN

NOTIZEN

III-12-12 Kv 2030/85